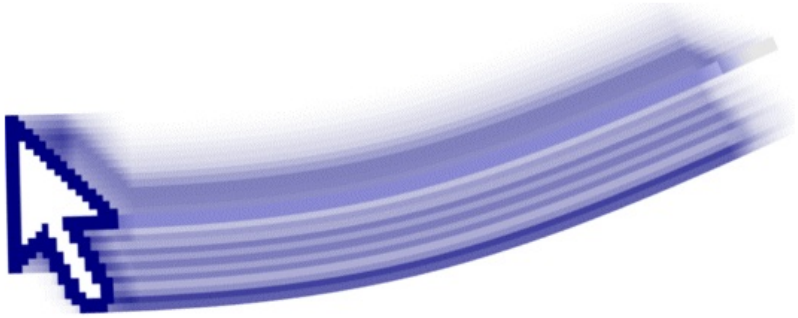
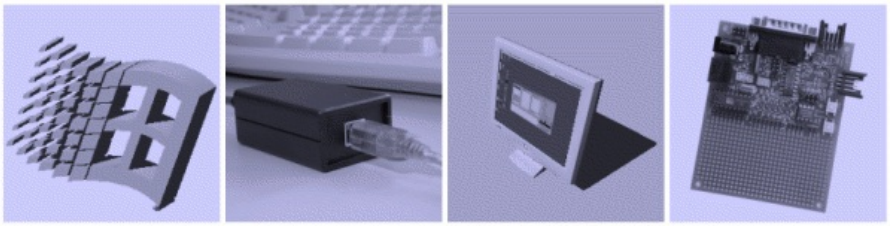
telos I2C Studio 5.5
User Manual





telos Systementwicklung GmbH
Schlueterstrasse 16, 20146 Hamburg /Germany
Phone: +49 (0)40 450 173 61
Mail: support-i2cstudio@telos.info
Web: www.telos.info

July 29, 2014

Contents

1 Terms 7

2 Introduction 9
- 2.1 Disclaimer . . . 9
- 2.2 Product Outline and Purpose . . . 9
  - 2.2.1 I2C Studio . . . 9
  - 2.2.2 I2C Framework . . . 10
  - 2.2.3 I2C Flasher . . . 10

3 Installation 11
- 3.1 System Prerequisites . . . 11
- 3.2 Software Installation . . . 11
- 3.3 Driver Installation . . . 11
- 3.4 telos Tracii 400 Installation . . . 13
- 3.5 Mac OS X . . . 13

4 Getting Started 15
- 4.1 I2C Studio Overview . . . 15
- 4.2 Function Windows . . . 15
- 4.3 Workspace . . . 17
- 4.4 I2C Scheduler . . . 18
- 4.5 Hardware Options . . . 20
- 4.6 I2C Interface "Dummy" . . . 21

5 Tracer 23
- 5.1 First Steps . . . 23
  - 5.1.1 Creating a new Tracer Window . . . 23
  - 5.1.2 Views . . . 23
  - 5.1.3 Acquire I2C Data . . . 25
- 5.2 Configuration . . . 26
  - 5.2.1 Tracer Options . . . 26
  - 5.2.2 IRD & Plug-In Manager . . . 26
- 5.3 Analog Mode . . . 29
  - 5.3.1 Introduction . . . 29
  - 5.3.2 Acquire Analog Data . . . 29
  - 5.3.3 Configuration . . . 29
- 5.4 Time View . . . 34
  - 5.4.1 Frame Data . . . 34
  - 5.4.2 Analog Data . . . 35
  - 5.4.3 Digital Input Lines . . . 36

5.4.4 Navigation . . . 36
5.4.5 Zoom . . . 36
5.4.6 Measuring . . . 37
5.4.7 Export / Copy to Clipboard . . . 37
5.5 Message View . . . 40
5.5.1 Navigation . . . 40
5.5.2 Raw Mode . . . 41
5.5.3 Register Mode . . . 44
5.5.4 Value Mode . . . 46
5.5.5 Export / Copy to Clipboard . . . 47
5.6 Live View . . . 50
5.7 Statistic View . . . 53
5.7.1 Statistic . . . 53
5.7.2 Export / Copy to Clipboard . . . 54
5.8 Find . . . 57
5.8.1 Find Window . . . 57
5.8.2 Rules . . . 58
5.9 Storage of Trace Data . . . 68
5.10 Printing . . . 69
5.10.1 Printer Configuration . . . 69
5.10.2 Data Selection . . . 70
5.11 Plug-Ins . . . 72

6 Master **75**
6.1 Introduction . . . 75
6.2 Creation of Messages . . . 75
6.2.1 Command Tree . . . 75
6.2.2 I2C . . . 77
6.2.3 SMBus . . . 80
6.2.4 IRD . . . 81
6.2.5 Pause . . . 81
6.2.6 Input Testpins . . . 82
6.2.7 Output Testpins . . . 83
6.3 Results . . . 83
6.3.1 I2C . . . 84
6.3.2 SMBus . . . 85
6.3.3 IRD . . . 86
6.3.4 Pause . . . 86
6.3.5 Input Testpins . . . 87
6.3.6 Output Testpins . . . 87
6.4 Configuration . . . 88
6.4.1 Master Options . . . 88
6.4.2 IRD & Plug-In Manager . . . 89

7 Negative Tester **91**
7.1 Introduction . . . 91
7.2 Test Cases . . . 93
7.2.1 I2C Master . . . 93
7.2.2 I2C Slave . . . 96
7.3 Options . . . 96
7.4 Result . . . 98

8 Scripting **101**
8.1 Introduction . . . 101
8.2 Example . . . 101
8.3 Configuration . . . 104
8.4 Advanced Examples . . . 104
8.5 I2C.NET API (IRD) . . . 105

9 Tools **107**
9.1 I2C Memory Slave . . . 107
9.2 VESA DDC . . . 108
9.3 JEDEC SPD . . . 108
9.4 SMBus Address Resolution Protocol . . . 110
9.5 IRD Compiler . . . 112
9.6 IRD Composer . . . 113

10 License & Support **115**
10.1 License Management . . . 115
10.1.1 Service Contracts . . . 115
10.1.2 Upgrade Features . . . 115
10.1.3 Registration Dialog . . . 115
10.2 Support . . . 116
10.3 Newsletter . . . 117

11 Redistribution **119**
11.1 Overview . . . 119
11.2 Driver . . . 119
11.3 I2C Scheduler . . . 119
11.4 API . . . 119
11.4.1 C . . . 120
11.4.2 C++ . . . 120
11.4.3 .NET . . . 120
11.4.4 Java . . . 120
11.4.5 Labview . . . 121

1 Terms

Combined Message	A sequence of submessages where only the last submessage is terminated by a STOP condition
telos Connii Interface	USB I2C interface for master operations.
telos Connii MM Interface	USB I2C interface for master and limited tracer operations. Successor of the telos Connii interface.
telos Connii MM 2.0 Interface	USB I2C interface for master and limited tracer operations. Successor of the telos Connii MM interface.
EPP I2C Interface	I2C interface which is connected to the PC via the parallel port. (The term EPP I2C interface applies only to telos products.)
GUI	Graphical User Interface
I2C Interface	Generic term for telos Tracii XL 2.0, telos Tracii XL, telos Tracii 400, telos Connii MM 2.0, telos Connii MM, and telos Connii interfaces.
Message	An I2C transfer that begins with a START condition and ends with a STOP condition. A message consists of one or more submessages without a STOP condition in-between.
Submessage	An I2C transfer that begins with a START condition and ends with a STOP condition or with a subsequent START condition.
telos Installation CD	CD which is delivered with the product. It contains all software and documentation which is needed to use the product.
telos Tracii 400 Interface	EPP I2C interface for master and tracer operations.
telos Tracii XL Interface	USB I2C interface for master, tracer, and oscilloscope operations.
telos Tracii XL 2.0 Interface	USB I2C interface for master, tracer, and oscilloscope operations. Successor of the telos Tracii XL interface.
USB I2C Interface	I2C interface which is connected to the PC via the Universal Serial Bus (USB). (The term USB I2C interface applies only to telos products.)

2 Introduction

The following sections are an introduction to the application I2C Studio.

2.1 Disclaimer

This product is not designed for use in life support appliances, devices or systems where malfunction of this product can reasonably be expected to result in personal injury.

Customers using or selling this product for use in such applications do so at their own risk and agree to fully indemnify telos for any damages resulting from such improper use or sale.

2.2 Product Outline and Purpose

telos I2C Studio is an easy-to-use integrated I2C development environment to access all features of the I2C interfaces in the telos product chain.

It supports usage of the USB clients telos Tracii XL 2.0, telos Tracii XL, telos Connii MM 2.0, telos Connii MM, and telos Connii as well as the parallel port device telos Tracii 400.

The installation package contains the following components: I2C Studio, I2C Framework, and I2C Flasher.

2.2.1 I2C Studio

I2C Studio is the GUI to access all features of the I2C interfaces in the telos product chain.

The main functionalities offered by I2C Studio are master, tracer, and negative tester. An additional feature of the tracer is an optional oscilloscope capability to give developers the possibility for watching the analog signal form in parallel to the traced I2C data. I2C Studio also supports scripting in C# for advanced users.

The I2C Studio has integrated an I2C slave, which works as RAM emulation and is only usable with telos Tracii XL 2.0 and telos Tracii XL.

Furthermore, it offers tools for JEDEC SPD RAM and VESA DDC support. See chapter 9 for more details.

The telos I2C Studio allows accessing multiple functionalities on a referenced I2C interface at the same time. This means that users can access one I2C interface e.g. as master and tracer at the same time. Thus, it is possible to send and trace I2C messages within one application simultaneously.

The application offers a modern graphical user interface based on the Microsoft .NET technology, which is highly configurable due to the usage of workspaces. Different view modes allow analyzing the traced I2C data regarding different aspects.

A special view model is part of the tracer and allows watching the data on datasheet level. It is called IRD. Instead of working with addresses, offsets and byte values the user is provided

with a comprehensive view of device registers, which are given by the names used in the datasheets.

2.2.2 I2C Framework

The I2C Framework consists of application programming interfaces (APIs) to communicate with telos I2C products within customer applications. The following APIs are part of the framework:

- .NET API
- C++ API
- C API
- Java API
- Labview API
- URT/URD API

2.2.3 I2C Flasher

The I2C flasher is an industrial solution for EEPROM and flash programming needs. It supports a bunch of I2C memories from different vendors as Atmel, Catalyst, Fairchild, ISSI, Microchip, NXP (Philips), Samsung and ST Microelectronics.

3 Installation

The following sections describe the installation of the hardware and the software.

3.1 System Prerequisites

The following preconditions are required for a computer system to install and use I2C Studio:

Memory	Minimum 512 MB RAM, 1 GB are recommended
Processor	Pentium IV 2.4 GHz or equivalent
OS Platforms	Windows XP (SP2)
	Windows Vista (x86 & x64)
	Windows 7 (x86 & x64)
	Windows 8 (x86 & x64)
	Windows 8.1 (x86 & x64)
Browser	Internet Explorer, version 8 or higher
Software Packages	Microsoft .NET-Framework 4.0
	telos Tracii 400 Install Package (optional)

3.2 Software Installation

To start the installation execute the file `Setup.exe` located in the directory `.\software\windows` on the telos setup CD.

Please note that administrator rights are required to install the software. The installation wizard leads the user through the installation process. Besides the main application the installation routine also installs the USB device driver for the telos I2C interfaces to the installation directory.

3.3 Driver Installation

After I2C Studio is installed properly, Microsoft Windows has drivers available for all telos I2C USB interfaces.

Now, if a new telos USB I2C interface is plugged into an USB port, the Microsoft Windows hardware wizard comes up and guides the user through a fresh driver installation for this specific device.

The wizard asks whether it should install the driver automatically or manually. Please choose the automatic installation. Now the wizard updates the driver installation with the newly found device. It is ready to use, then.

Note: Microsoft Windows also starts the driver installation process, if a telos USB I2C device has formerly been in use on the system but the USB port has changed.

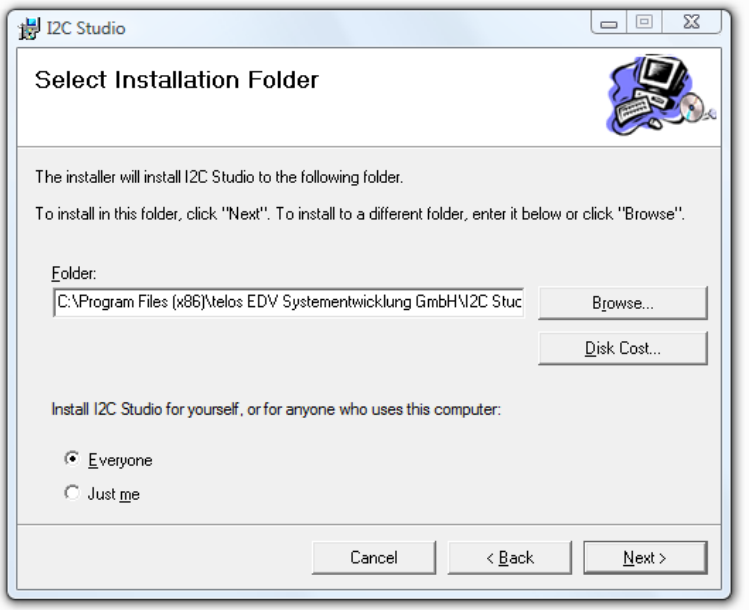

I2C Studio
Select Installation Folder
The installer will install I2C Studio to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files (x86)\telos EDV Systementwicklung GmbH\I2C Stuc
Browse...
Disk Cost...
Install I2C Studio for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >

Figure 3.1: I2C Studio Install Wizard

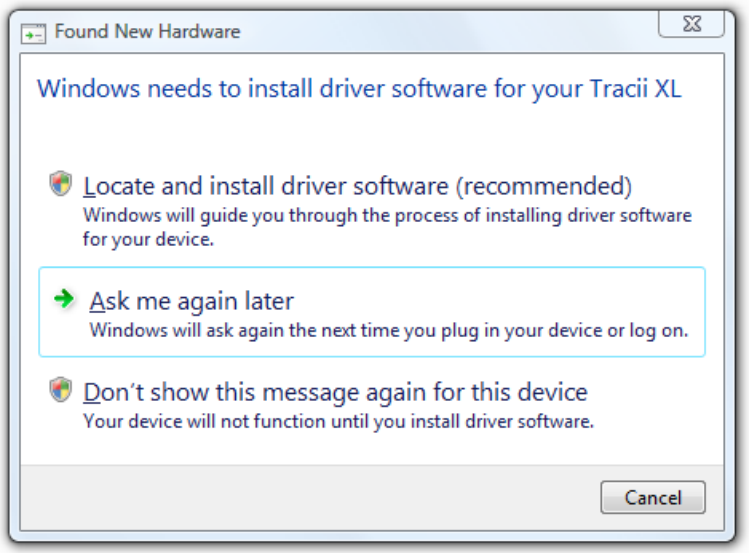

Found New Hardware
Windows needs to install driver software for your Tracii XL
Locate and install driver software (recommended)
Windows will guide you through the process of installing driver software for your device.
Ask me again later
Windows will ask again the next time you plug in your device or log on.
Don't show this message again for this device
Your device will not function until you install driver software.
Cancel

Figure 3.2: New Hardware Wizard

3.4 telos Tracii 400 Installation

I2C Studio supports the operation of Tracii 400 interfaces. Due to the fact that telos Tracii 400 is a parallel port device it requires some additional installation steps to use it:

- The telos Tracii 400 device driver must be installed manually.
- I2C Studio has to be set up to support telos Tracii 400 interfaces.

The telos Tracii 400 driver is stored in a separate package on the telos installation CD. Besides the driver the package contains the applications TraciiSpy and TraciiTalk. Both items are not required and should therefore be deselected during installation process.

Executing the program `tracii400\Disk1\Setup.exe` from the telos installation CD starts the installation package.

Before a telos Tracii 400 interface can be accessed by the telos I2C Studio, the parallel port being used has to be selected and after that, the interface's license key has to be assigned.

The following steps should be followed to prepare a telos Tracii 400 interface:

- The telos Tracii 400 interface has to be attached to the desired LPT port.
- telos Tracii 400 support must be enabled in I2C Studio (see 4.4).
- The license key delivered by telos must be entered using the menu File|Registration (see chapter 10).

3.5 Mac OS X

There is no native Mac OS X application for interfacing the telos I2C tools. However, telos I2C Studio has been tested to perform well in a virtual machine running Windows 7.

What you need:

- An Intel Mac built 2010 or later - earlier models may work but lack performance. I2C Studio will not run on Power PC.
- VM Fusion virtualization software available from:

  `http://www.vmware.com`

- telos I2C Studio
- Microsoft Windows 7 (later versions may work too)

Please follow these steps to install your telos I2C interface:

1. Install VM Fusion on your Mac.
2. Install MS windows by either doing a fresh install or by using the VMWare migration assistant to move your existing installation over to the Mac.
3. Please uninstall telos I2C studio, if you migrated an existing windows installation, which had this software installed.

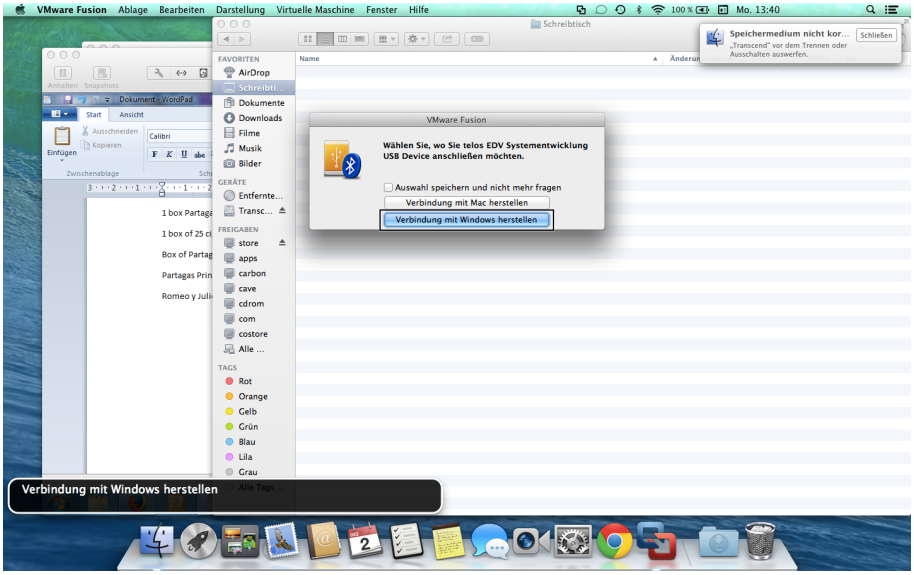

VMware Fusion
Wählen Sie, wo Sie telos EDV Systementwicklung USB Device anschließen möchten.
Auswahl speichern und nicht mehr fragen
Verbindung mit Mac herstellen
Verbindung mit Windows herstellen
Verbindung mit Windows herstellen

Figure 3.3: Mac OS X

4. Do not connect any I2C equipment yet.

5. Run the telos I2C Studio setup wizard. You should see the drivers being installed and you should be prompted to confirm the driver installation.

6. After the telos I2C Studio setup wizard has completed, connect your telos I2C interface to the USB port of your Mac while Windows is running in VMWare fusion. You should see the dialog shown in figure 3.3.

7. The dialog prompts you in your localized language to chose between connecting the equipment to the Mac or to your virtual Windows machine. Please chose to connect to the Windows machine and tick the checkbox to make this selection permanent.

8. You are done, telos I2C studio should now interact properly with your hardware.

You may use the shared folder facility of VM Fusion to directly store log files created by telos I2C Studio to your native Mac file system. Folder shares are usually placed on your desktop by VMWare.

4 Getting Started

4.1 I2C Studio Overview

Starting telos I2C Studio the first time the user in confronted with a splash screen for a while. Depending on the system speed, it takes a moment until I2C Studio opens up with a blank workspace.

What the user is presented starting I2C Studio the first time is a nearly blank workspace on the left side and a so-called start page on the right.

The applications window top hosts a menu bar and a toolbar with some buttons beneath. Most of the toolbar buttons and parts of the menu bar selections are greyed out at the very first start simply because no function window has been opened in I2C Studio so far.

In this state another component of the application is nearly empty: The status bar resides at the bottom of the application window, showing only a link to the telos homepage as its default setting.

4.2 Function Windows

To start working with I2C Studio it is necessary to create a function window. I2C Studio knows several different types of function windows: master, tracer, negative tester, and script. Not all I2C interfaces support all types of function windows.

	Master	**Tracer**	**Negative Tester**	**Script**
telos Tracii XL 2.0	X	X	-	X
telos Tracii XL	X	X	-	X
telos Connii MM 2.0	X	X	-	X
telos Connii MM	X	X	-	X
telos Connii	X	-	-	X
telos Negative Tester	-	-	X	X
telos Tracii 400	X	X	-	X

To create a new function window simply press on the New button. This will open the dialog shown in figure 4.2.

This dialog contains a tree with three levels. The first level contains the PC, to which the I2C interface is connected. The second level contains the I2C interface. In this example we have two PCs. One real I2C interface is connected to the local PC: a telos Tracii XL with the serial number 0xD4D8. A telos Connii with the serial number 0x1EA5 is connected to the remote PC with the DNS name dyn16-113.telos.de (IP address 194.173.126.113, IP port 3000). The third level of the tree contains the function window types supported by each I2C interface.

To open e.g. a tracer function window on the telos Tracii XL connected to the local PC simply double click on the Tracer line beneath telos Tracii XL. After double clicking a new function window is created. It gets displayed on the right side of I2C Studio.

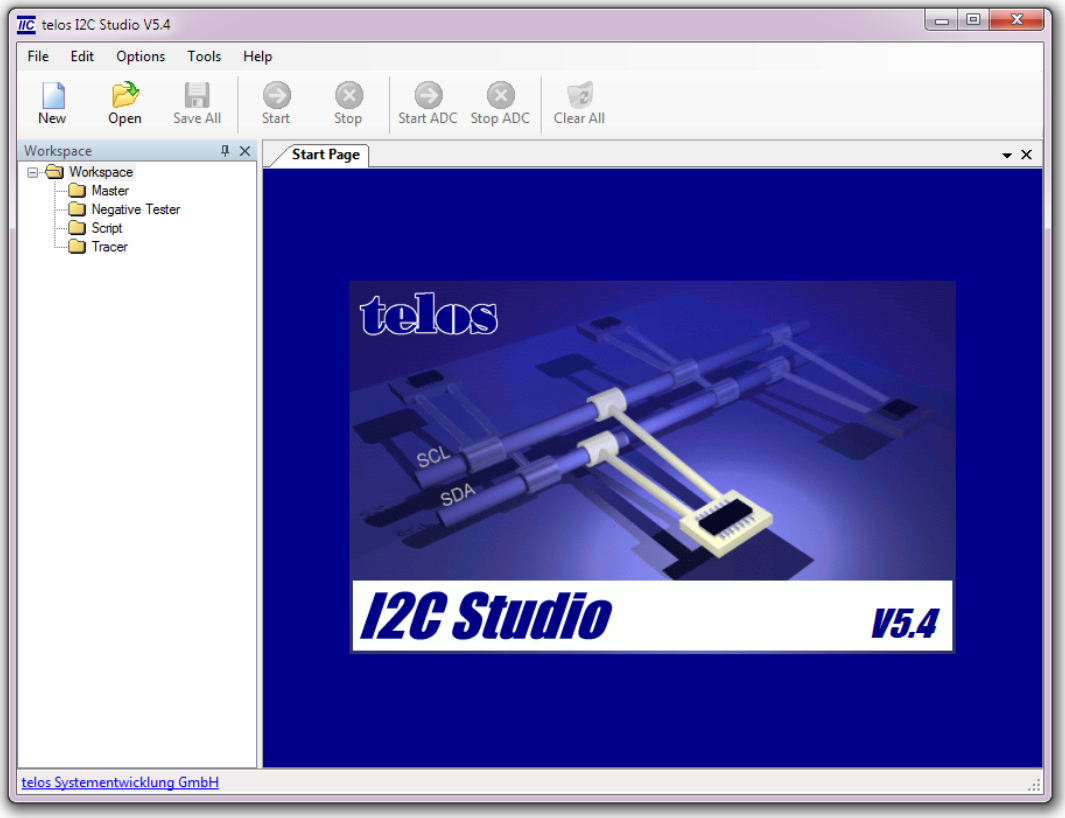


Figure 4.1: I2C Studio after Startup

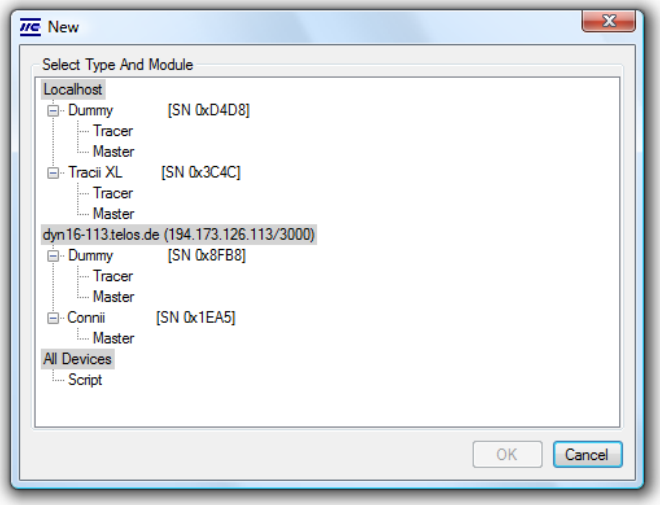


Figure 4.2: New Dialog

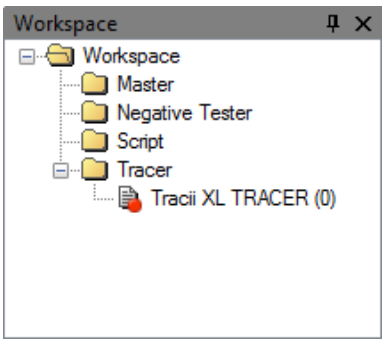


Figure 4.3: Workspace

To close a function window click on the X button on the upper right side of the window.

4.3 Workspace

After creating a new function window an entry is added to the tree of documents in the workspace window on the left side of I2C Studio, see figure 4.3.

The tree contains four folders, one for each type of function window supported by I2C Studio. In these folders I2C Studio stores links to opened and closed function windows.

The icons in the tree have got the following meaning:

Folder closed

Folder opened

Function window is closed

Function window is closed and cannot be opened

Function window is opened

Function window is opened (stopped state)

Function window is opened (running state)

The workspace, which is pictured in the example, contains one open function window. It is a function window for the tracer functionality of a telos Tracii XL interface. The tracer is in the stopped state, which can be determined by the icon.

When the user moves the cursor of the mouse over a document in the workspace some information about the document are shown within a tooltip. By double clicking on a closed document, it is possible to open the document.

According to the state of the function window the context menu of a document offers different options:

Open/Close Open or close the appropriate function window file.

Reconnect Associate a new I2C interface with the function window.

Remove From Disk Delete the function window file from disk and its link from the workspace.

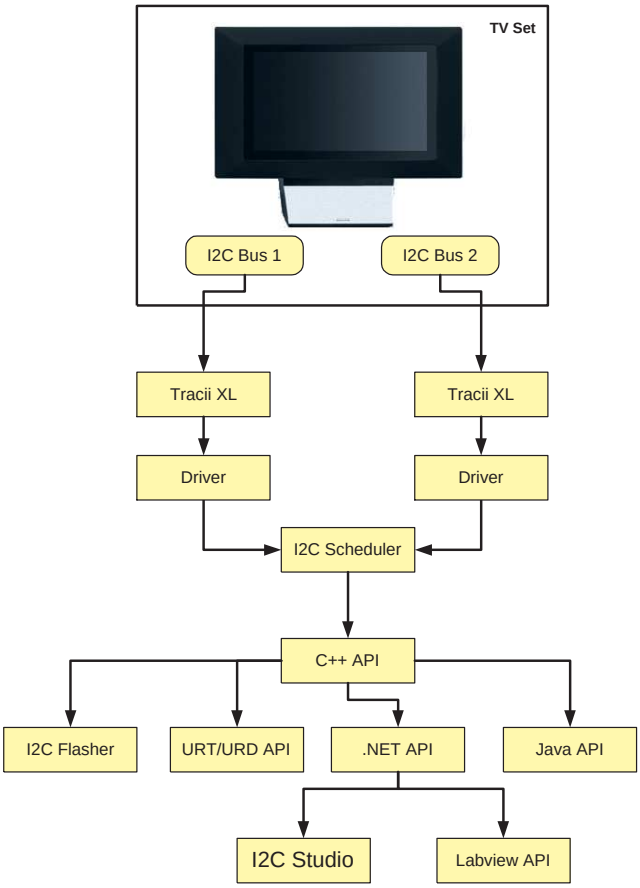


Figure 4.4: I2C Framework

Remove From Workspace Delete the function window from Workspace. The file is still available on disk.

Rename Change the name of the function window file.

It is possible to create a new function window not only by using the New button. Another way leads to the context menu of the workspace. Therefore the user simply opens the context menu on one of the folder icons in the tree of the workspace. To create e.g. a master function window, the user opens the context menu on the Master folder and selects New.

The configuration of the workspace is stored in files with the prefix `*.i2cw`. Workspace files can be created, opened and stored using the items in the File|Workspace menu.

4.4 I2C Scheduler

Figure 4.4 gives an overview of the I2C Framework and I2C Studio. One central component of the I2C Framework is the I2C scheduler. This component connects the I2C interfaces with

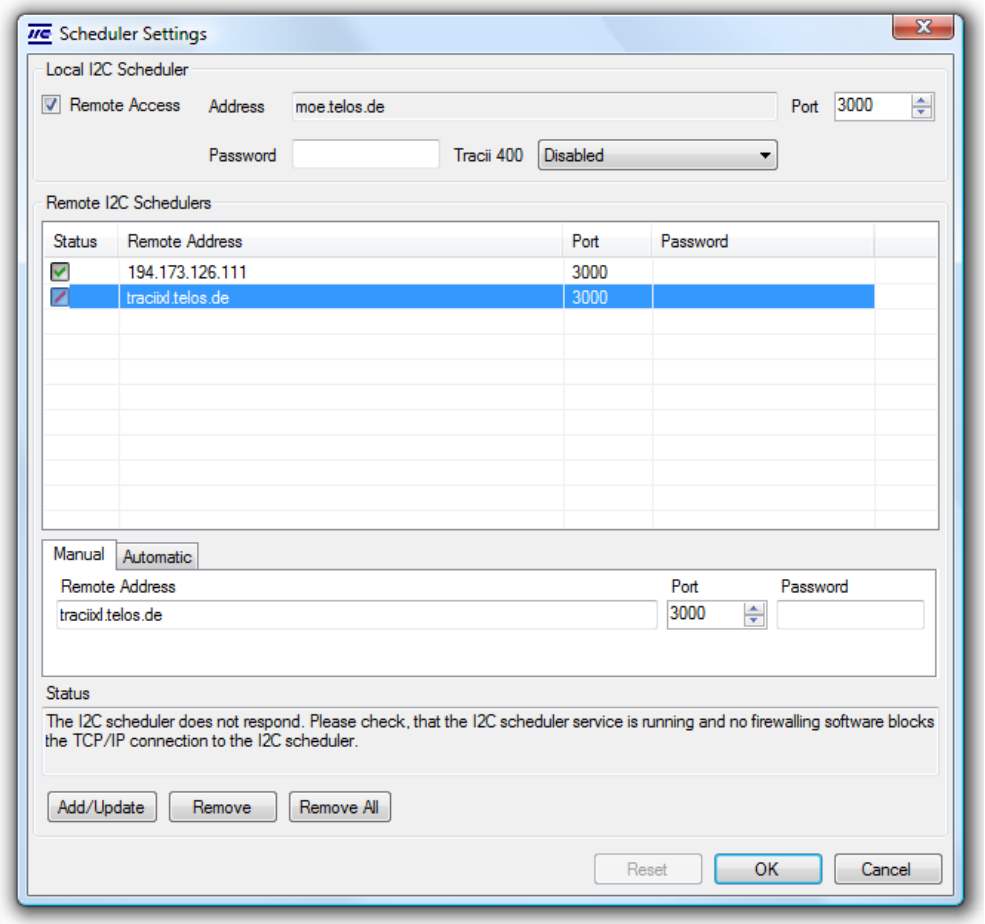


Figure 4.5: Scheduler Settings

the I2C APIs. It is implemented as MS Windows service, which gets started automatically at startup of the PC.

The I2C scheduler acts like a multiplexer/demultiplexer. It makes it possible to communicate simultaneously from several programs with the same I2C interface.

The communication between the I2C scheduler and the I2C APIs uses a TCP/IP channel. Due to this fact it is possible to communicate not only with I2C interface connected to the local PC but also with I2C interfaces connected to other PCs in the LAN. This can be useful e.g. for remote diagnostics.

I2C Studio offers a dialog to configure the I2C scheduler running on the local PC and the connection to I2C schedulers running on other PCs. This dialog can be opened by selecting Options|Scheduler Settings in the menu.

The dialog can be divided into two major parts. The upper part serves the configuration of the local I2C scheduler. The Remote Access check box configures whether the local I2C scheduler should accept connection from other PCs in the LAN. The Port input configures the TCP/IP port, on which the I2C scheduler is listing for incoming connections. To prevent a misusage of the local I2C scheduler, it is important to set a secure password.

If there is a telos Tracii 400 interface connected to the local PC, the Tracii 400 combo box can

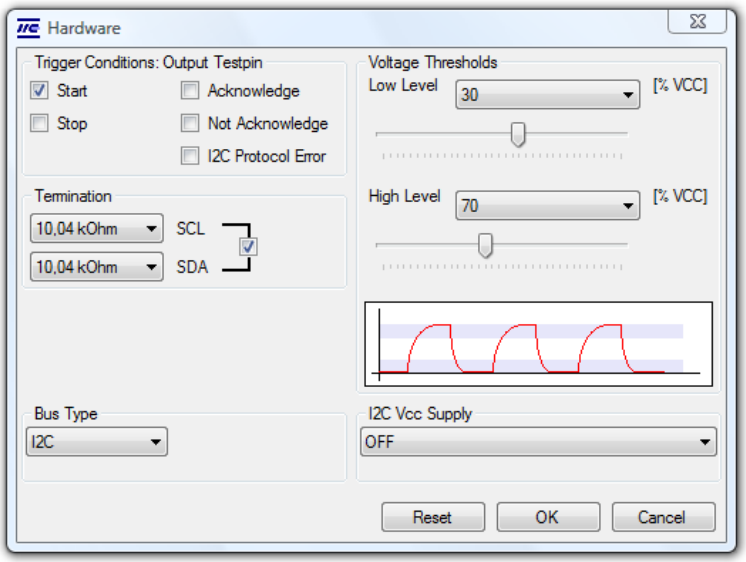


Figure 4.6: Hardware Options

be used to configure the parallel port used by the interface.

The second part of the dialog serves the configuration of the connections to I2C schedulers running on remote PCs. To connect to a remote I2C scheduler three parameters are needed: TCP/IP address (DNS name or IP address), TCP/IP port, and password.

In the example, which is shown in figure 4.5 two remote I2C schedulers have been specified. The Status column of the table shows the current status of the connection. In the example the connection to the first I2C scheduler has been established successfully. The second I2C scheduler cannot be reached by I2C Studio. By selecting one line in the table it is possible to get a detailed description of the reason, why a remote I2C scheduler cannot be contacted.

4.5 Hardware Options

The I2C interfaces offer several hardware and electrical parameters, which can be configured by the software.

Both the master and the tracer function window offer a dialog to configure these parameters. In the master function window this dialog can be opened using the Master|Hardware Options item in the menu. The same dialog can be opened in the tracer function window using Tracer|Hardware Options.

Trigger Conditions: Output Testpin Some I2C interfaces provide an output testpin that can signal certain events. E.g. the I2C interface can trigger this testpin, when it detects a START condition on the I2C bus.

Termination Dependent on the bus load the termination of the I2C bus can be adjusted to manipulate the signals slew rate. For long cables and/or high bitrates it is important to have a sufficient low termination of the I2C bus.

Voltage Threshold The voltage thresholds are parameters that define the interpretation of the analog signals. If an I2C bus signal is high and falls below the low level threshold, it is treated as a high/low transition. If an I2C bus signal is low and rises above the high level threshold, it is treated as a low/high transition. The default values for an I2C bus conforming to the I2C specification is a setting of 30%/70%.

Bus Type I2C Studio supports I2C buses (I2C) and SMBuses (SMB).

I2C Vcc Supply All I2C interfaces have got a I2C Vcc supply pin on their I2C connector. The level on this pin defines the high level of the I2C bus. Some newer I2C interfaces can generate this voltage internally. The internal supply can be configured using this option.

4.6 I2C Interface "Dummy"

The I2C scheduler does not only communicate with I2C interfaces connected to the local PC, but also implements a virtual I2C interface. This virtual I2C interface can be used e.g. to evaluate I2C Studio without real hardware connected to the PC. Or it can be useful for developing software based on the I2C Framework.

Connected to this virtual I2C interface are five virtual I2C devices:

Slave Address	Device
0x0B	Smart Battery: Battery
0x0C	Smart Battery: Charger
0x29	Analog Devices ADM1021A
0x50	256 bytes RAM
0x68	Dallas DS1307

Some virtual communication between a virtual I2C master and these devices is generated automatically. But it is also possible to communicate with these device manually using e.g. the master function window.

5 Tracer

One of the function window types, which can be opened in I2C Studio is the tracer window. In contrast to the master window this type of window does not work actively on the I2C bus. Instead it records the traffic between other I2C masters and slaves on the I2C bus passively.

Recording the data of an I2C bus can be useful in different scenarios. E.g. there is an existing system, which does not work as expected. The tracer can then be used as a debugging tool to find e.g. wrong or illegal I2C messages. Another scenario could be the reverse engineering of an existing system, to make e.g. an new system compatible to the existing one.

5.1 First Steps

5.1.1 Creating a new Tracer Window

To open a new tracer window the developer can use the New button, the File|New menu item or the context menu of the workspace.

Note: Not all I2C interfaces can be used as I2C tracer: some I2C interfaces cannot be used as I2C tracer at all, other require a special license.

Figure 5.1 shows how I2C Studio looks like after opening a new tracer function window. A new item has been added to the workspace. The new tracer function window is displayed on the right side.

5.1.2 Views

The tracer function window is divided into different views. These views offer different interpretations of the same data traced on an I2C bus:

Time View Visualizes the data in the time domain. This can be used to detect e.g. stretching on the I2C bus or to measure the speed of I2C masters on the I2C bus.

Message View Visualizes the data as a list of I2C messages. This view is useful, when only the transferred data without detailed timing information is of interest. Using this view it is not only possible to see the transferred data bytes, but it is also possible to get an I2C device depended interpretation of the data.

Live View Shows the current value of all registers of the devices connected to the traced I2C bus.

Statistic View Contains some statistics about the transfers on the I2C bus.

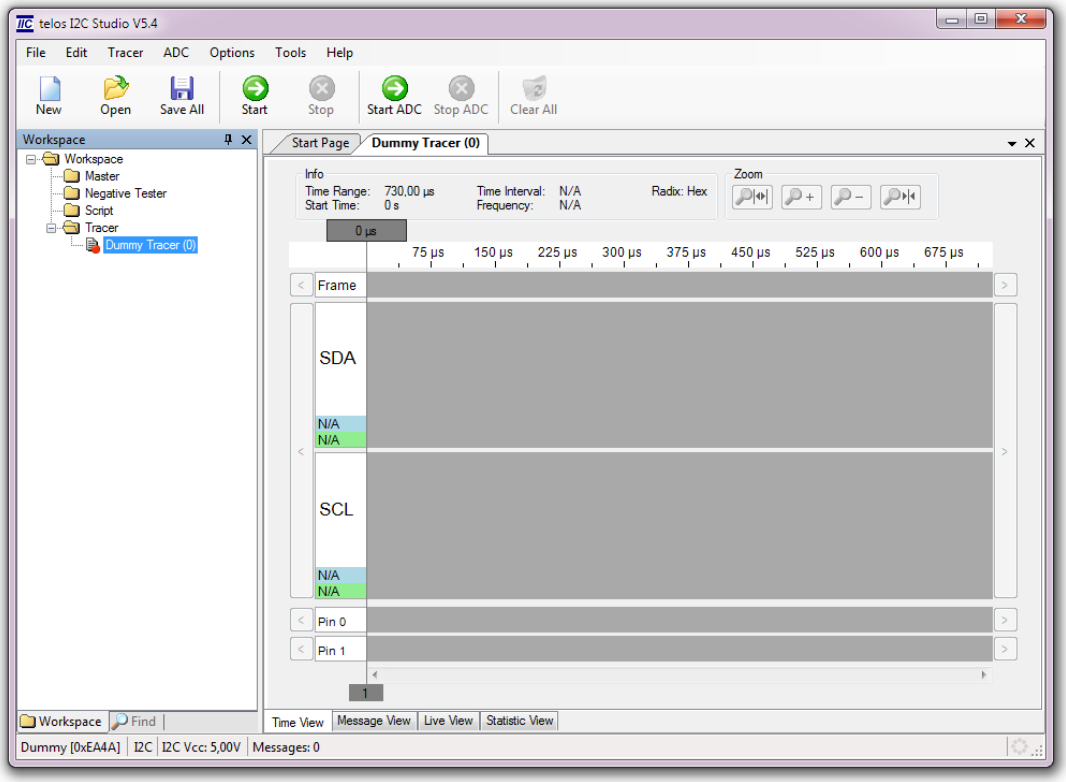


Figure 5.1: New Tracer Window

5.1.3 Acquire I2C Data

Before it is possible to use any of the views it is necessary to acquire I2C data. The acquisition can be started using the Start button or the menu item Tracer|Start. As soon as the tracer is running the Start button gets greyed out and the Stop button gets enabled.

If there is traffic on the I2C bus, the "Messages" value in the status bar gets incremented periodically. This value contains the number of I2C messages traced on the I2C bus. A message starts with a START condition and ends with a STOP condition. This means that combined I2C transfers consisting e.g. of two submessages will be counted as only one I2C message.

The traced I2C data gets streamed to the hard disc, so that the number of I2C messages, which can be traced, is limited only by the space available on the hard disc containing the Windows system directory for temporary files.

While the tracer is running most functions of the tracer function windows can be used in parallel. E.g. it is possible to use the different views to inspect the already traced data. Some functions can be used only while the tracer is in the stopped state.

To finish the acquisition of data the Stop button or the menu item Tracer|Stop can be used.

Restarting the tracer will not delete the already traced data. Instead the new data will be appended to the already acquired data. The deleting of the acquired data can be requested by using the Clear All button, which will also delete some other data, or by the menu item Tracer|Clear.

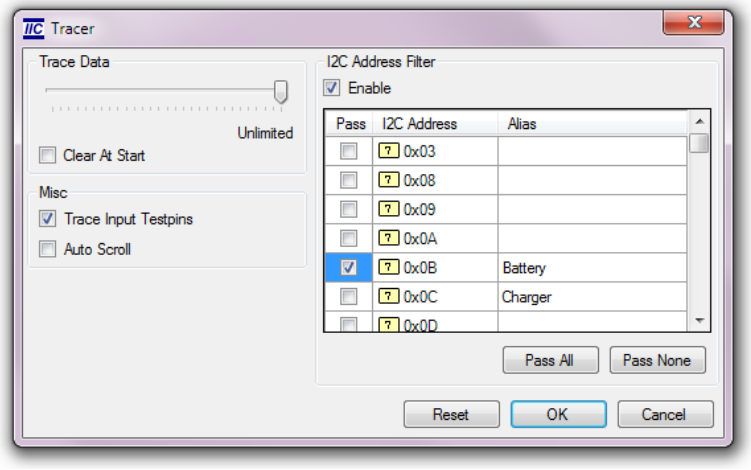


Figure 5.2: Tracer Options

5.2 Configuration

5.2.1 Tracer Options

Some properties of the tracer function window can be modified using the options dialog of the window. This dialog can be opened using the menu item Tracer|Tracer Options. Figure 5.2 shows the tracer options dialog.

In some situations it is necessary to let the tracer run for hours, but only a limited number of messages are of interest to debug a problem. For such situations I2C Studio offers the possibility to limit the data queue to a specific size. This limit can be configured using the slider in the "Trace Data" group box. If the limit is reached, the oldest messages are removed.

If the "Clear At Start" check box is set, the already captured data gets deleted when the tracer is restarted.

Some I2C interfaces have additional digital input lines, whose state can be traced in parallel to the I2C bus. If the check box "Trace Input Testpins" is set, these lines are traced.

The check box "Auto Scroll" defines, whether the view should jump to the end of the data automatically, when new data arrives. This can be useful especially on an I2C bus, on which not too many messages are transferred. If this option is enabled, I2C Studio will consume a lot of more resources during tracing. So this option should be used with caution.

Sometimes only messages to specific I2C slaves are of interest. In such situations the I2C address filter can be used. If it has been enabled, only messages to slaves whose I2C addresses are marked as "pass" get recorded. All other messages are not recorded. This means it is not possible to make them visible again at a later time.

5.2.2 IRD & Plug-In Manager

Beside the hardware and the tracer option dialogs there is a third dialog under the Tracer menu item: the "IRD & Plug-In Manager" dialog.

This dialog, which can be seen in figure 5.3, has got three tasks:

- assign alias names to I2C addresses

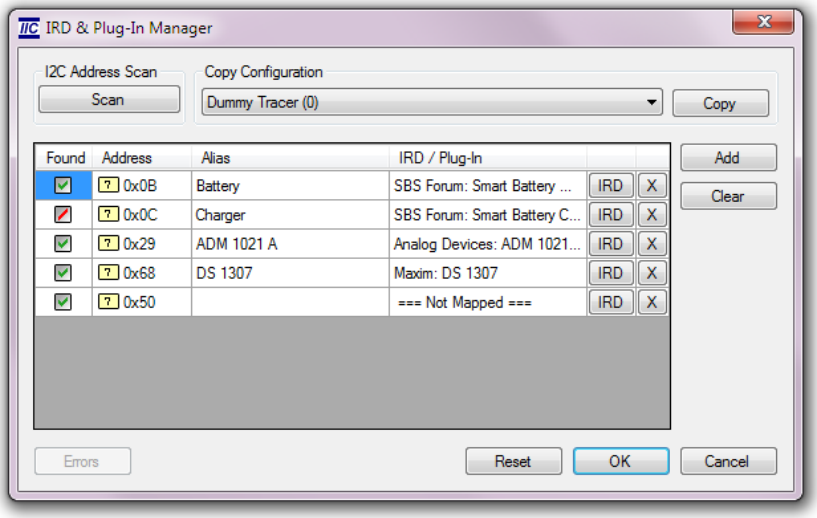


Figure 5.3: IRD & Plug-In Manager

- assign IRD or plug-in files to I2C addresses
- execute a scan for slaves connected to the I2C bus

Using the Copy button it is possible to copy the configuration of another open function window.

Alias Names

For a human being is it always difficult to remember which I2C devices uses which I2C address. So to make it more convenient it is possible to assign an alias name to an I2C address. The tracer function window will show these alias names additionally to the I2C addresses in all places, where I2C addresses are used.

To add a new alias name simply press on the Add button. A new form will be opened asking for the I2C address. After entering the desired I2C address, the user can simply click somewhere outside the form. This will add the I2C address to the table and close the form. In the table the user can click into the ”Alias” cell of the new row and enter the desired alias name.

IRD/Plug-Ins

When debugging complex problems it is very time-consuming to compare the transferred raw data with the data-sheets of the used ICs. That is why I2C Studio offers the I2C Register Description (IRD) and the plug-in interface.

Using IRD or the plug-in files I2C Studio can display the raw data in an interpreted format. This higher-level view displays the meaning of the transferred data for an IC. This means that e.g. I2C Studio displays that ”20 °C” has been read from the ”External Temperature” register of a temperature sensor.

IRD is a XML based file format, which can be used to describe the register and values of an IC. The plug-ins can be written with any programming language supporting Microsoft’s

.NET framework. I2C Studio comes along with a large number of ready to use IRD and plug-in files. In addition the user can write its own files.

To assign an IRD or plug-in file to an I2C address, simply open the "IRD/Plug-In" combo box in the appropriate row of the table and choose the desired IRD or plug-in file. If there is not already a row for the desired I2C address, use the Add button.

User written IRD and plug-in files must be copied to the following folders:

IRD	`<Documents>\I2C Studio\ird`
Plug-In	`<Documents>\I2c Studio\plugins`

I2C Address Scan

When working with an existing system, it is not always known, which I2C addresses are used by the I2C slaves. To examine an I2C bus an I2C address scan can be started by pressing the Scan button.

The address scan tries to read data from each valid I2C address. If the read is answered, this I2C address gets marked as to be in use.

Note: The I2C address scan does not necessarily find all I2C addresses, which are in use. A lot of I2C devices on the market do not comply exactly to the I2C specification. Furthermore the I2C slave can be in a busy state, while the I2C address scan is running.

The result of the I2C address scan is shown in the first column of the table. The icons have got the following meaning:

- I2C address is in use
- I2C address is not in use or no I2C address scan has been run

5.3 Analog Mode

5.3.1 Introduction

The normal I2C tracer is a great help to debug most of the daily problems when working with I2C busses. In some cases the digital view of the I2C bus gives the developer not enough information to fix his problems.

In such situations the analog mode of the tracer comes into play.

Note: Not all I2C interfaces support the analog mode: some I2C interfaces does not support this feature at all, other require a special license.

The analog mode works like a stand alone oscilloscope. The user specifies a trigger condition and some other parameters and starts the analog mode. As soon as the trigger condition gets fulfilled by the data transferred on the I2C bus, the I2C interface starts to record analog samples of the SCL/SDA lines. Like with a stand alone oscilloscope the sample depth is limited by the sample buffer on the I2C interface:

I2C Interface	Sample Depth
telos Tracii XL	64 kSamples/line
telos Tracii XL 2.0	256 kSamples/line

After the sampling has been completed, the data gets transferred to the PC. When the transfer is completed, the analog mode can be restarted

I2C Studio visualizes the analog data in parallel to the I2C messages and the levels of the input testpins.

5.3.2 Acquire Analog Data

The waiting for the trigger can be activated by using the menu item ADC|Start or by using the Start ADC button in the toolbar. After selecting start the Start ADC button gets greyed out and the Stop ADC button gets enabled.

When the analog mode has found the trigger condition on the I2C bus and the analog data has been transferred to the PC, the Start ADC button gets activated again.

Starting the acquisition of the analog mode again will not delete the samples of the previous acquisitions. There are two ways to delete the acquired analog data. The Clear All will delete the samples together with the traced I2C messages and the levels of the input testpins. The menu item ADC|Clear will delete only the analog data.

In some situations trigger conditions do not really help to find a problem, e.g. if there the lines of the I2C bus have not been connected correctly. For such situations I2C Studio offers the manual shot, which can be started using the menu item ADC|Manual Start. This starts the sampling of the analog data immediately.

5.3.3 Configuration

There are a lot of options, which can be configured for the analog mode. The configuration dialog can be opened by clicking on the ADC|Options menu item. Figure 5.4 shows the configuration dialog.

The following parameters can be configured:

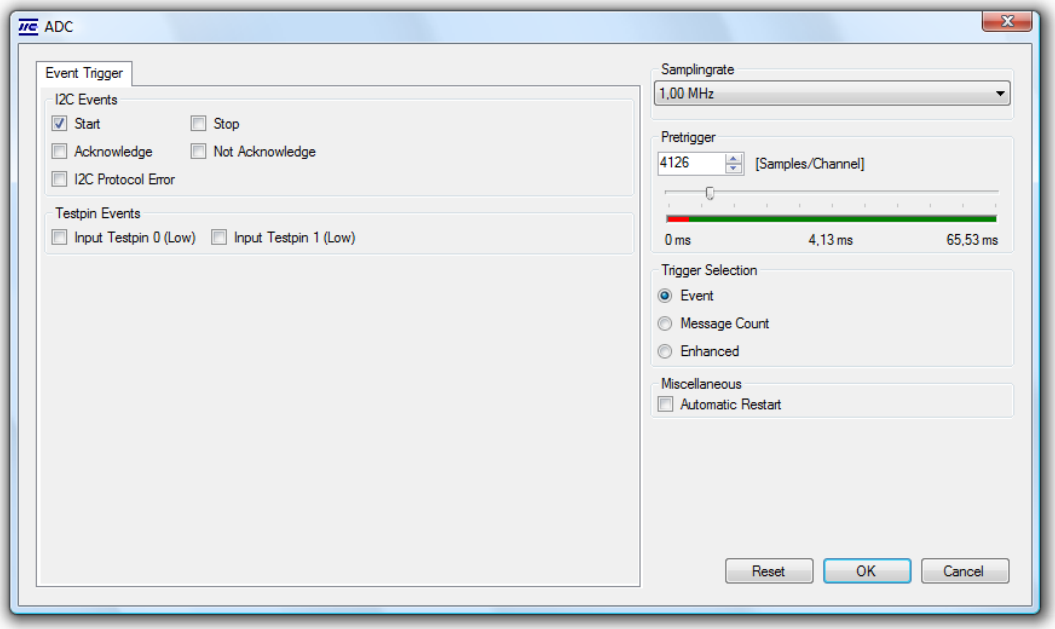


Figure 5.4: ADC Options: Event Trigger

Sampling Rate Defines how many samples per second get recorded. Because the sample depths of the I2C interfaces are limited, this parameter also defines the length of the shot. Typically the sampling rate should be at least at factor 10 higher than the highest I2C clock rate on the I2C bus. So if the system under test uses I2C clock rates up to 100 kHz, the sampling rate should be at least 1 MHz to get a good resolution of the recorded signal.

Pretrigger For most scenarios the developer is interested not only in the analog waveform after the trigger condition, but also some samples before the trigger condition are needed. The pretrigger value defines, how many samples should be recorded prior and after the trigger condition.

Trigger Selection I2C Studio offers three groups of ADC trigger conditions. The active one can be selected using this option.

Miscellaneous Normally the ADC module takes only one shot after receiving the trigger condition. If the ”Automatic Restart” switch is enabled, I2C Studio waits automatically for the next trigger after the previous ADC shot has been completed.

Note: There is always a short time between finishing the sampling of one shot and reactivating the trigger module for the next one. Due to this fact I2C Studio may miss some trigger conditions.

When the user changes the ”Trigger Selection”, the left part of the dialog gets changed.

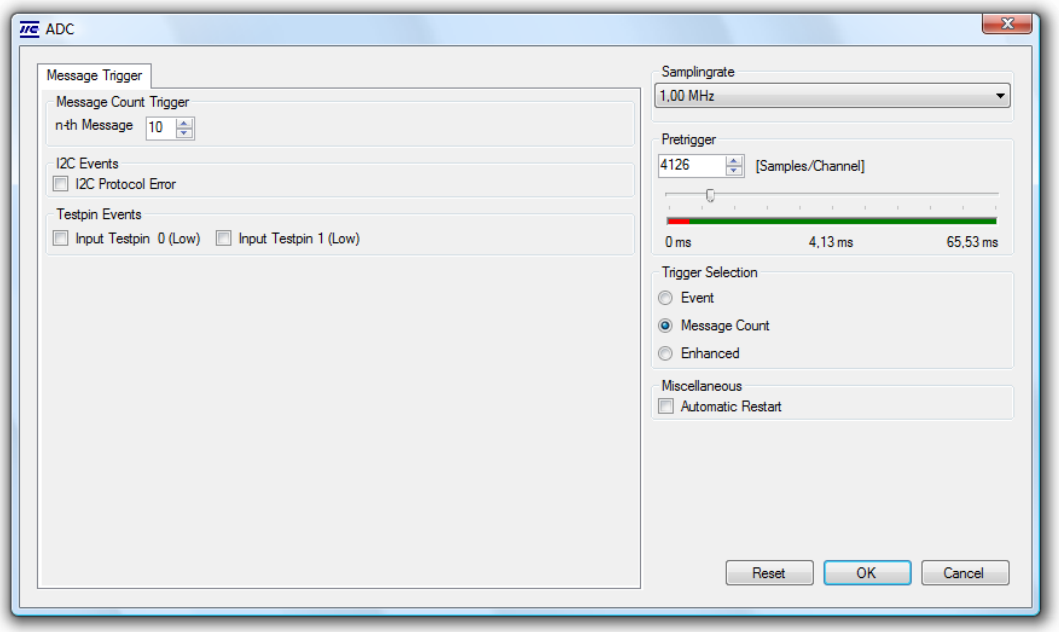


Figure 5.5: ADC Options: Message Trigger

Event Trigger

The event trigger group contains very basic trigger conditions. It is possible to trigger on the following I2C events:

- START condition
- STOP condition
- acknowledge
- not acknowledge
- I2C protocol error

If more than one condition is enabled, the ADC shot gets triggered when one of the enabled conditions is fulfilled.

These triggers are very easy to configure and use, but on an I2C bus with heavy load they are only of limited use, because e.g. normally all I2C messages start with a START condition.

Besides the trigger on I2C events this group also contains triggers for the input testpins. A high-low transition on an input testpin can be used as trigger condition.

Message Trigger

In some cases the developer knows exactly at which position the message of interest is sent. In such a scenario the ”Message Count Trigger” can be used. If this value is set e.g. to ”10”, the shot is triggered with the 10th message seen on the I2C bus, after pressing the Start ADC button.

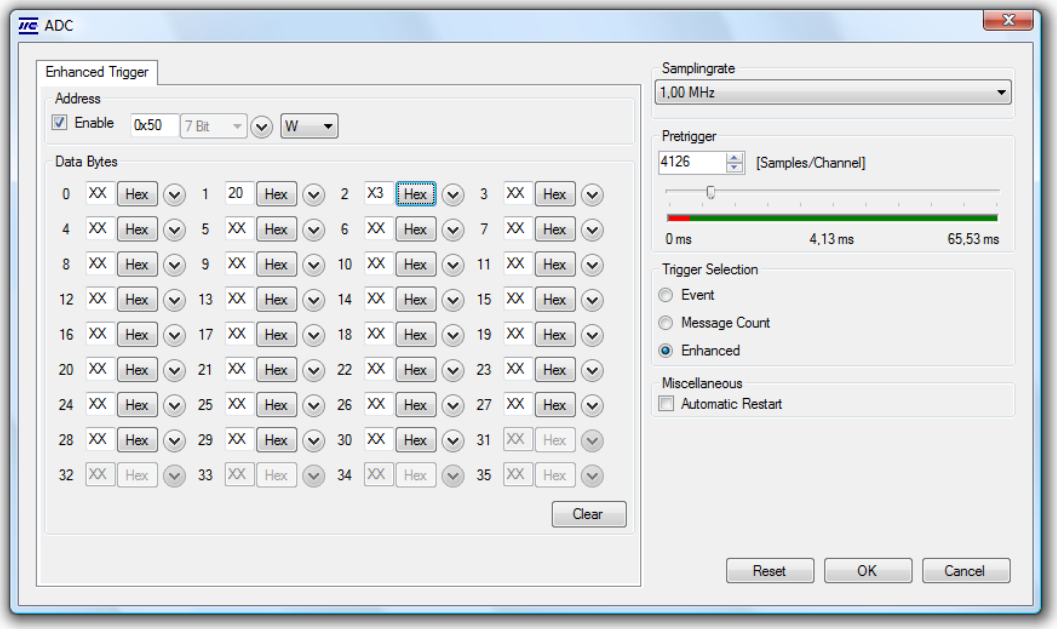


Figure 5.6: ADC Options: Enhanced Trigger

Additional it is possible to trigger on an ”I2C protocol error” and on high-low transitions of the input testpins.

After one of the enabled trigger conditions get fulfilled, the ADC shot is triggered.

Enhanced Trigger

On an I2C bus with several devices connected to complex trigger conditions are needed. In such scenarios the ”Enhanced Trigger” mode could be used, which can be seen in figure 5.6.

Using the ”Address” field it is possible to trigger on I2C messages, which are sent to a device with the specified slave address. Furthermore it is possible to configure the direction of the transfer to be triggered on:

W The master writes data to the slave.

R The master reads data from the slave.

R/W The master reads or writes data from/to the slave.

The ”Data Bytes” field can be used to trigger on specific values for data bytes. E.g. the dialog in figure 5.6 has been configured to trigger, when the second byte of an I2C submessage equals ”0x20” and the lower 4 bits of the third byte equals ”0x03”.

All values in the ”Address” and the ”Data Bytes” field must be fulfilled in the same I2C submessage to trigger the ADC shot.

To give a small example of the ”Enhanced Trigger” usage lets assume the following setup: there is I2C slave at address 0x30, whose high-level protocol is register-based. The registers are addressed by a 8-bit register pointer. The user wants to get an analog shot of the time when the I2C master reads data from register ”0x55”.

For this scenario the user would set the ”Address” field to ”0x30” and ”W”. In the ”Data Byte” field the user would set the first byte to ”0x55” and all other bytes to ”Don’t Care” (XX).

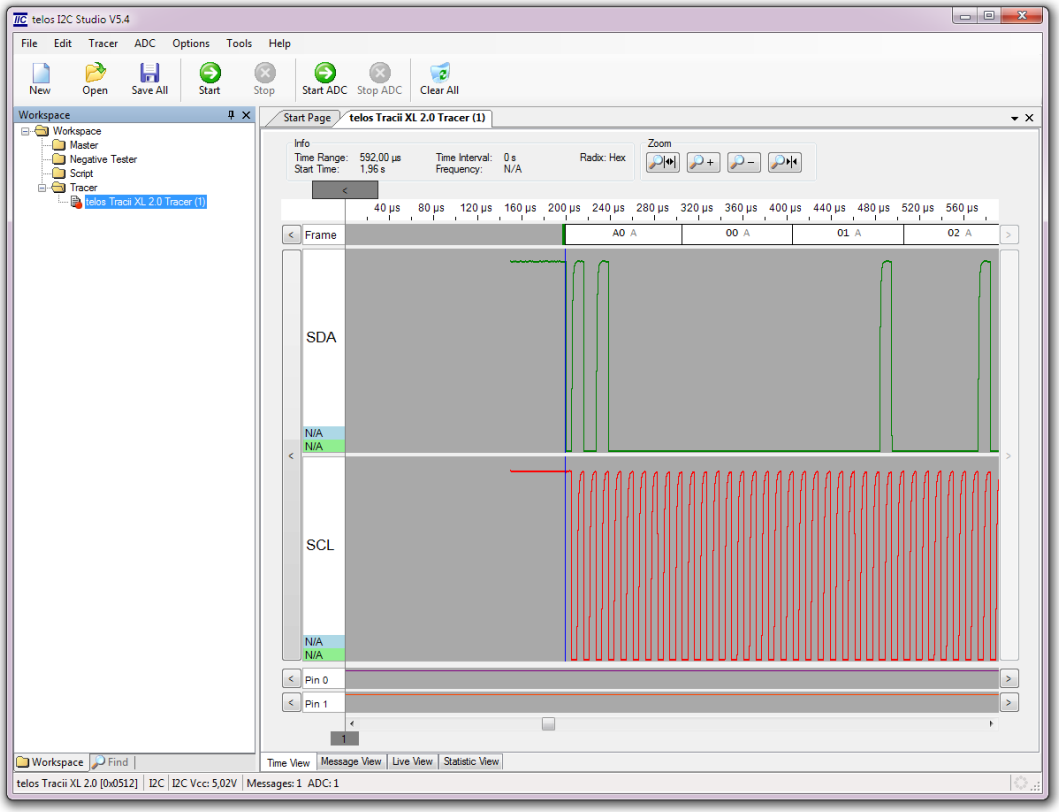


Figure 5.7: Time View

5.4 Time View

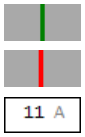
The time view offers a visualization of the traced data in the time domain. Figure 5.7 shows the time view.

The visualization can be divided into three parts: frame data, analog data, and input testpin data. All parts are synchronized.

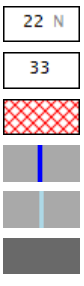
Note: The timing information of the I2C interfaces ”telos Connii MM”, ”telos Connii MM 2.0”, and ”telos Tracii 400” are not accurate. Due to this reason the time view gives only a rough information of the timing on the I2C bus for these I2C interfaces.

5.4.1 Frame Data

The frame data part displays the data, which has been recorded on the I2C bus. The following symbol are used for the visualization:



START condition
STOP condition
data byte with ACK



data byte with NACK
data byte with missing ACK/NACK
I2C protocol error
SMBus power off
SMBus power on
tracer was not running

It is possible to change the radix of the displayed data bytes using the context menu.

Using the context menu switch Virtual Bit Level (see figure 5.8) it is possible to enable a second interpretation of the I2C data. This view shows the high/low transitions of the SCL/SDA lines.

Note: The "Virtual Bit Level" view is not based on timing information measured on the I2C bus. The timing of the real transitions on the I2C bus may differ from the one in the visualization.

If the option Messages Tool Tips in the context menu is enabled, additional information about an I2C message are shown, when the mouse cursor is moved over the elements of this I2C message.

5.4.2 Analog Data

The analog data part shows the analog electrical levels on the I2C lines SDA and SCL.

During the development process the developer often needs to have a look at the electrical layer of the I2C bus. The I2C bus has got several parameters like e.g. the bus termination or the cable capacitance, which the I2C bus has to fulfil to get a working set-up.

Adjusting these parameters is much easier, if the user has got a view of the electrical levels. Standard digital oscilloscopes come to their limitation, if they are used for the I2C bus, because most of them cannot trigger on I2C conditions.

The switches in the context menu under Analog can be used to enable or disable the view of the SCL and/or SDA analog data.

The position of the trigger in the analog data gets marked with a blue line.

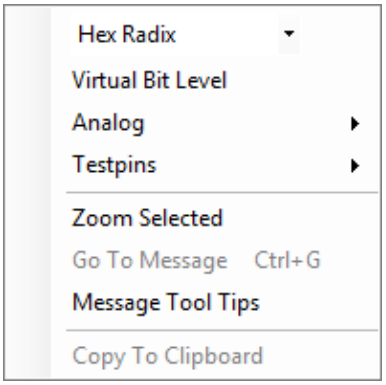


Figure 5.8: Time View: Context Menu

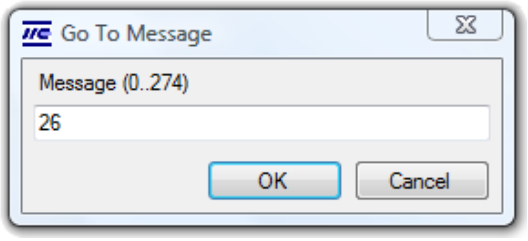


Figure 5.9: Time View: Go To Message

5.4.3 Digital Input Lines

The transitions on additional digital input lines (input testpins) can be monitored in the third part of the time view.

The number of the available lines depend on the used I2C interface.

If the bus type has been set to SMBus for the tracer function window, the lines get renamed from ”Pin 0” to ”Alert” and from ”Pin 1” to ”Suspend”. This reflects the special meaning of the lines for the SMBus.

The visualization of the input testpins can be enabled or disabled using the context menu items under Testpins.

5.4.4 Navigation

There are various methods to navigate within the time view.

One method is to simply use the vertical scroll bar. With this scroll bar it is possible to move forward and backward in the traced I2C data. This is a good method to navigate within the data bytes of one message, but it can be quite difficult to find e.g. the beginning of the next I2C message.

In such a situation another method comes into play. There are several $<$ and $>$ buttons. They have got the following meaning in the different parts of the time view:

Frame Data Jump to the previous or next START condition.

Analog Data Jump to the previous or next beginning of an analog shot.

Digital Input Lines Jump to the previous or next transition of the line.

To jump to an I2C message with a known number, the ”Go To Message” dialog can be used, which is shown in figure 5.9. This dialog can be opened using the context menu item Go To Message.

By double-clicking somewhere into the data all other views are synchronized to this time.

5.4.5 Zoom

At the upper right corner of the time view there are four buttons to zoom in the visualization. The buttons have got the following meaning:

There is an additional way to zoom into the data. By holding the left button of the mouse down while moving the mouse over the time view it is possible to select a part of the currently visualized data. Using Zoom Selected from the context menu this part can be magnified to the full width of the time view window.

5.4.6 Measuring

The time view offers two cursors, which can be used to measure the signals on the SDA and SCL lines. The cursors can be moved by holding down the left mouse button over one of the cursors.

The upper box of each cursor displays the time difference between the left border of the time view and the cursor. The time corresponds to the time ruler on the upper part of the time view.

If there is analog data available under the current position of the cursor, the corresponding voltages of the SDA and SCL line are displayed on the left side of the time view.

Some more time information can be found in the "Info" box at the upper part of the time view. The values of this box have got the following meaning:

Time Range Describes the length of time of the data currently displayed starting from the left side and ending with the right side of the time view.

Start Time Time of the left edge of the time view measured from the time, when the tracer has been started.

Time Interval The time between the two cursors.

Frequency "Time Interval" converted to a frequency value. This can be used to determine e.g. the frequency of an I2C master by measuring the bits of an analog shot.

Radix Radix currently used by the time view. It can be changed using the context menu.

5.4.7 Export / Copy to Clipboard

The visualization of the time view can be exported to an image. This image can be stored in an external file or in the system's clipboard.

To export the image to a file simply use the menu item File|Trace Data|Export. This opens the wizard for exporting the recorded I2C data. On the first page of the wizard the user must select the type of view which should be exported to the file. After choosing "Time View" a second page gets opened, which is used to configure what data should be exported, see figure 5.10.

If the image should be used in another MS Windows application, it can be exported directly to the clipboard of the system. This can be done by selecting Copy to Clipboard from the context menu of the time view. This opens directly the options page of the export wizard.

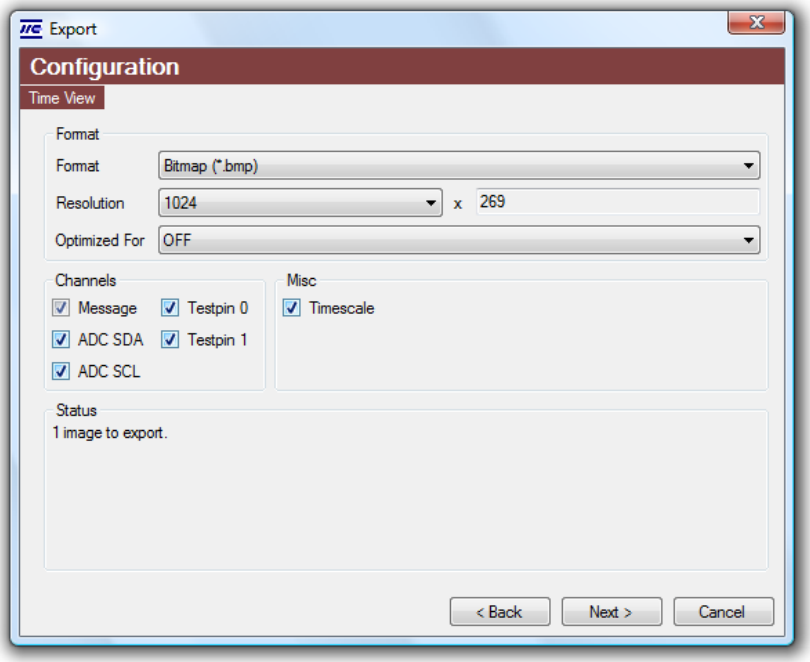


Figure 5.10: Time View: Export

The wizard will export those data, which is currently visible in the time view. If the user wants to export the complete traced data, he must zoom out so that everything is visible in the view and start the wizard.

The option page offers the following parameters, which can be configured:

Format The user can choose between the following three image formats:

Bitmap (*.bmp) Very common format in the MS Windows world, which is supported by nearly all programs. As these files are not compressed, they get quite large.

Joint Photographic Experts Group (*.jpg) Format offering a very high, but lossy compression. Has been developed for the storage of photographs, so that it does not offer a really good image quality for I2C Studio's purposes.

Portable Network Graphics (*.png) Offers a good lossless compression resulting in acceptable file sizes. Very good suitable for files created with I2C Studio.

Tagged Image File Format (*.tif) Offers a good lossless compression resulting in acceptable file sizes. Very good suitable for files created with I2C Studio.

Resolution Defines the width and the height of the images to be created. The user can only specify the width, the height gets calculated automatically.

Optimized For If the user selects to export several I2C messages at the same time, he will always result into images, whose resolution is too low to see any interesting details. To solve this problem, I2C Studio offers a functionality to split the result into several separate image files.

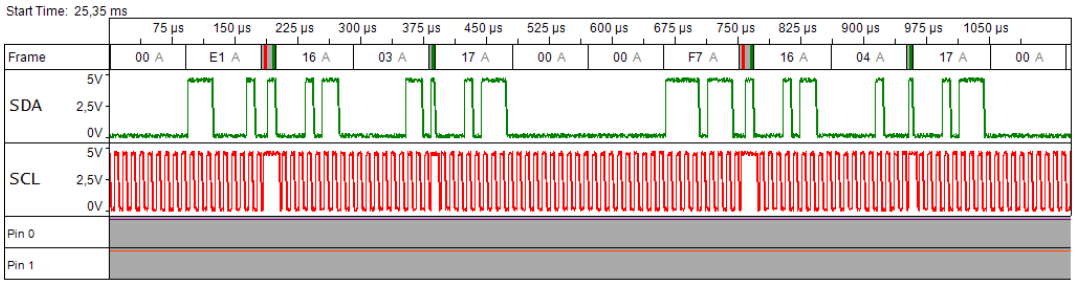


Figure 5.11: Time View: Exported Data

This option configures the number of images, which are created. To make the user the decision, how many images are needed, very easy, he can simply specify the needed resolution as frequency.

Let us assume the traced I2C bus uses a clock rate of 400 kHz. For this set-up the user would simply select an ”Optimized For” value of 400 kHz. If there are analog shots of the electrical levels available, typically higher frequencies like e.g. 6 MHz should be selected.

Channels Sometimes the user do not want to export all available parts of the time view. Using these switches the miscellaneous parts can be selected or deselected to be exported.

Misc The ”Timescale” options defines, whether a timescale is added to the exported files or not.

Status Displays the number of images, which will be created, when user would export the data with the current settings.

After clicking on Next > the third page of the wizard gets visible. On this page the user must enter the name of the file to be created. A click on the Finish button finally starts the creation of the file(s). Figure 5.11 shows a typical file, which has been created using the export wizard.

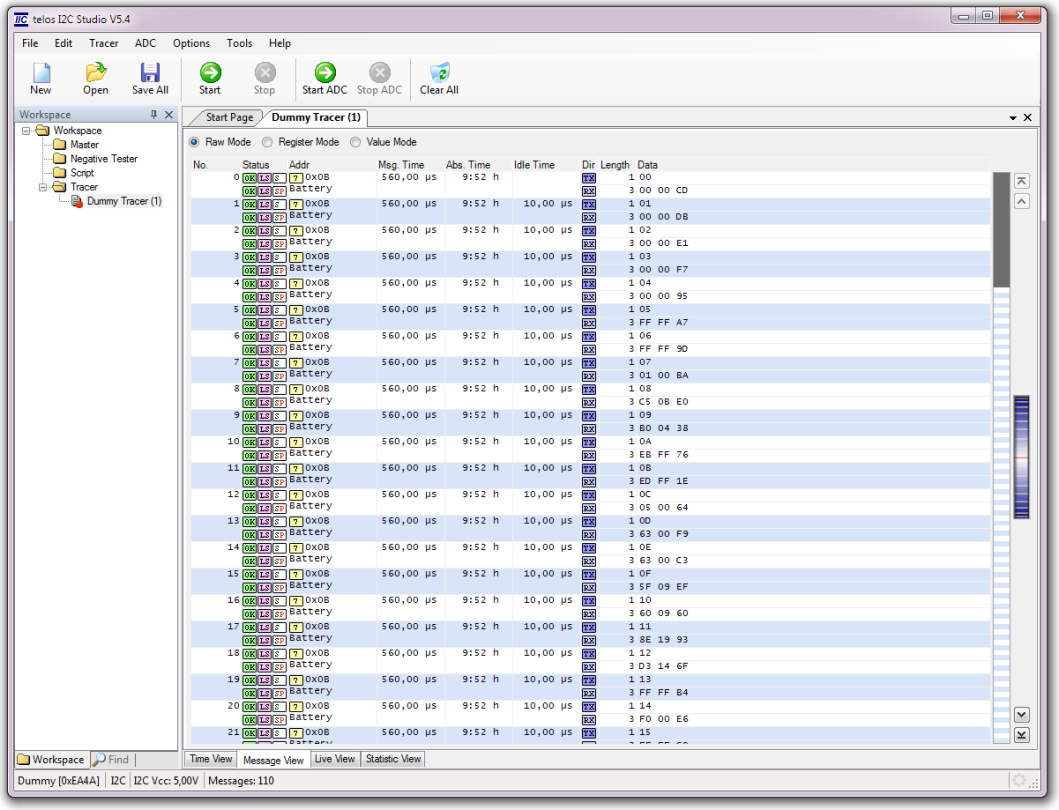


Figure 5.12: Message View: Raw Mode

5.5 Message View

Very often during the development process the exact timing information of each byte are not of interest. Instead the developer needs information about the transferred data. This is the domain of the message view in I2C Studio.

There are three different modes in the message view: raw mode, register mode, and value mode. It is possible to switch between these modes by using the toggle buttons at the upper part of the message view.

Figure 5.12 shows the message view in the raw mode. The I2C data, which has been traced on the I2C bus, gets displayed as a table.

5.5.1 Navigation

On the right border of the view there are some controls to navigate within the table.

The buttons have got the following functions:

- ⤒ Jump to the first message.
- ⌃ Jump to the previous message.
- ⌄ Jump to the next message.
- ⤓ Jump to the last message.

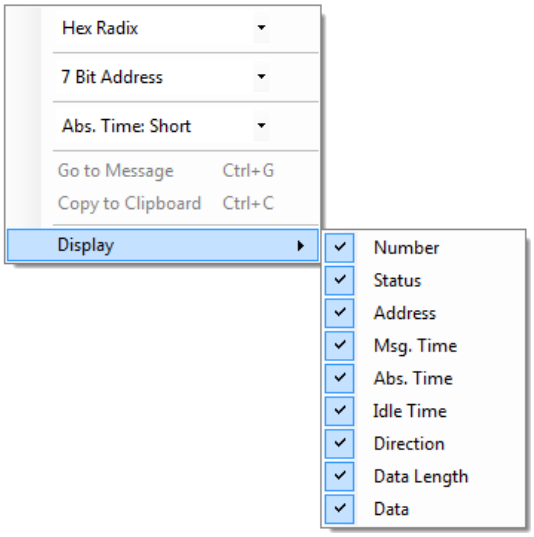


Figure 5.13: Message View: Context Menu

Another method to navigate in the I2C data is the thumb wheel. By moving the wheel up or down the message view starts to scroll upwards or downwards. After releasing the wheel, the scrolling stops. Scrolling speed is affected by the angle of the wheel.

Next to the buttons and the thumb wheel there is another control, which displays the current position within all I2C messages. This control can be used not only to determine the position but also to jump to a specific position. The user can simply click into the control to define the new position, which should be visualized by the message view.

To jump to an I2C message with a known number, the "Go To Message" dialog can be used, which is shown in figure 5.9. This dialog can be opened using the context menu item Go to Message.

By double-clicking on a row containing an I2C message all other views get synchronized to this message.

5.5.2 Raw Mode

The raw mode displays the data bytes as they get transmitted on the I2C bus.

Each row of the table contains one I2C message. If an I2C message consists of several I2C submessages, they are displayed among each other.

The columns of the table can be enabled or disabled using the context menu items under Display, see figure 5.13. They contain the following information:

No Position of the I2C message within the traced messages.

Status The three icons have got the following meaning:

1st Icon Shows whether the I2C messages has been transferred successfully:

no error

I2C address has not been acknowledged

one of the data bytes has not been acknowledged

2nd Icon Shows whether the I2C message has been transferred in the high-speed mode:

LS	I2C normal-speed
HS	I2C high-speed

3rd Icon Shows whether the START and the STOP conditions have been send one the I2C bus:

S	only START condition
SP	START and STOP condition
P	only STOP condition

Addr Address of the I2C slaves, which has received the I2C message. As default the address gets displayed as a 7-bit addresses with a hexadecimal representation. The numeric format can be changed using the context menu item Hex Radix. Some data sheets specify the I2C addresses in the 8-bit format. This format can be enabled in the message view with the 7 Bit Address switch in the context menu.

The icons can have got the following meaning:

7	7-bit I2C address
8	7-bit I2C address (8-bit representation)
10	10-bit I2C address

If an alias name has been registered for the displayed I2C address, this name gets shown in this column.

Msg. Time The message time is measured from the first START condition of the I2C message to the STOP condition.

Abs. Time The absolute time is the time of the first START condition. The user can choose between three different formats using the context menu item Abs. Time: Short. The formats "Short" and "Complete" refer to the real-time clock of the PC. In the format "Relative" the time is relative to the time, when the I2C tracer has been enabled the first time.

Idle Time The idle time is measured from the STOP condition of the previous I2C message to the START condition of the current I2C message.

Dir An I2C message can be transferred into two directions. This column displays, which one has been used:

TX	master transmitter
RX	master receiver

Length Number of data bytes (does not include the bytes of the I2C address).

Data Data bytes (does not include the bytes of the I2C address). Using the context menu it is possible to change the numeric format from hexadecimal to decimal.

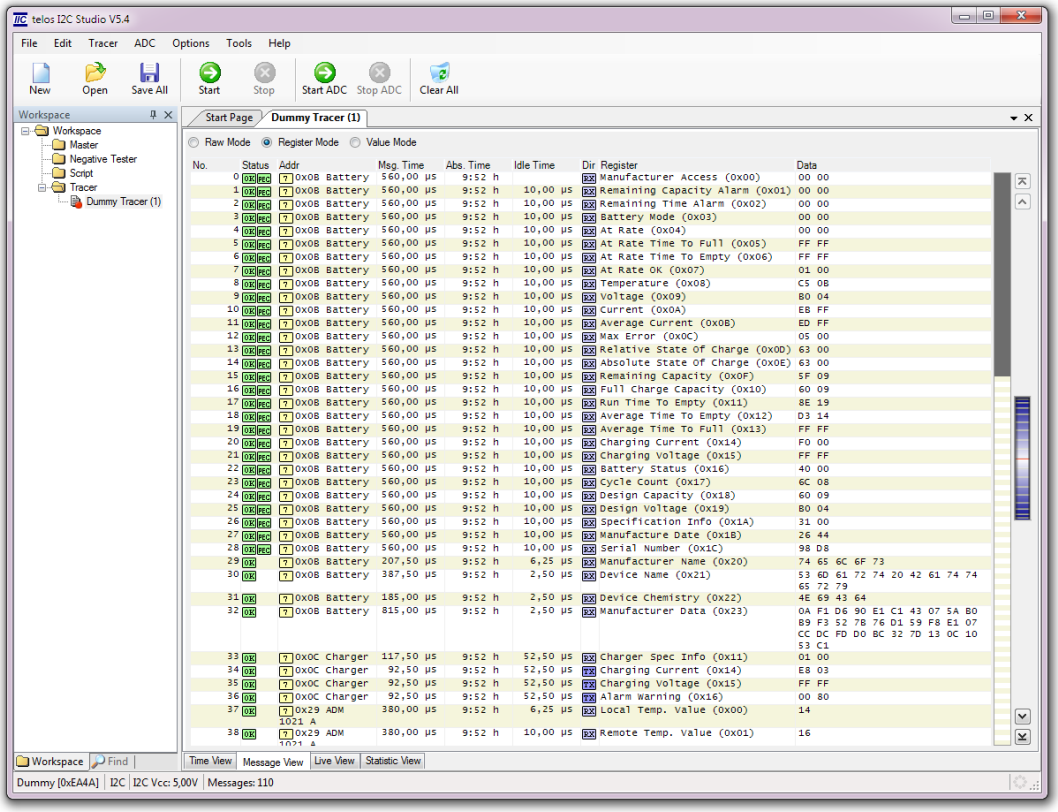

telos I2C Studio V5.4
File Edit Tracer ADC Options Tools Help
New Open Save All Start Stop Start ADC Stop ADC Clear All
Workspace
Workspace
Master
Negative Tester
Script
Tracer
Dummy Tracer (1)
Start Page
Dummy Tracer (1)
Raw Mode
Register Mode
Value Mode
No. Status Addr Msg. Time Abs. Time Idle Time Dir Register Data
0 0x0B Battery 560,00 µs 9:52 h Manufacturer Access (0x00) 00 00
1 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Capacity Alarm (0x01) 00 00
2 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Time Alarm (0x02) 00 00
3 0x0B Battery 560,00 µs 9:52 h 10,00 µs Battery Mode (0x03) 00 00
4 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate (0x04) 00 00
5 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate Time To Full (0x05) FF FF
6 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate Time To Empty (0x06) FF FF
7 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate OK (0x07) 01 00
8 0x0B Battery 560,00 µs 9:52 h 10,00 µs Temperature (0x08) C5 0B
9 0x0B Battery 560,00 µs 9:52 h 10,00 µs Voltage (0x09) B0 04
10 0x0B Battery 560,00 µs 9:52 h 10,00 µs Current (0x0A) EB FF
11 0x0B Battery 560,00 µs 9:52 h 10,00 µs Average Current (0x0B) ED FF
12 0x0B Battery 560,00 µs 9:52 h 10,00 µs Max Error (0x0C) 05 00
13 0x0B Battery 560,00 µs 9:52 h 10,00 µs Relative State Of Charge (0x0D) 63 00
14 0x0B Battery 560,00 µs 9:52 h 10,00 µs Absolute State Of Charge (0x0E) 63 00
15 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Capacity (0x0F) 5F 09
16 0x0B Battery 560,00 µs 9:52 h 10,00 µs Full Charge Capacity (0x10) 60 09
17 0x0B Battery 560,00 µs 9:52 h 10,00 µs Run Time To Empty (0x11) 8E 19
18 0x0B Battery 560,00 µs 9:52 h 10,00 µs Average Time To Empty (0x12) D3 14
19 0x0B Battery 560,00 µs 9:52 h 10,00 µs Average Time To Full (0x13) FF FF
20 0x0B Battery 560,00 µs 9:52 h 10,00 µs Charging Current (0x14) F0 00
21 0x0B Battery 560,00 µs 9:52 h 10,00 µs Charging Voltage (0x15) FF FF
22 0x0B Battery 560,00 µs 9:52 h 10,00 µs Battery Status (0x16) 40 00
23 0x0B Battery 560,00 µs 9:52 h 10,00 µs Cycle Count (0x17) 6C 08
24 0x0B Battery 560,00 µs 9:52 h 10,00 µs Design Capacity (0x18) 60 09
25 0x0B Battery 560,00 µs 9:52 h 10,00 µs Design Voltage (0x19) B0 04
26 0x0B Battery 560,00 µs 9:52 h 10,00 µs Specification Info (0x1A) 31 00
27 0x0B Battery 560,00 µs 9:52 h 10,00 µs Manufacture Date (0x1B) 26 44
28 0x0B Battery 560,00 µs 9:52 h 10,00 µs Serial Number (0x1C) 98 D8
29 0x0B Battery 207,50 µs 9:52 h 6,25 µs Manufacturer Name (0x20) 74 65 6C 6F 73
30 0x0B Battery 387,50 µs 9:52 h 2,50 µs Device Name (0x21) 53 6D 61 72 74 20 42 61 74 74 65 72 79
31 0x0B Battery 185,00 µs 9:52 h 2,50 µs Device Chemistry (0x22) 4E 69 43 64
32 0x0B Battery 815,00 µs 9:52 h 2,50 µs Manufacturer Data (0x23) 0A F1 D6 90 E1 C1 43 07 5A B0 B9 F3 52 78 76 D1 59 F8 E1 07 CC DC FD D0 BC 32 7D 13 0C 10 53 C1
33 0x0C Charger 117,50 µs 9:52 h 52,50 µs Charger Spec Info (0x11) 01 00
34 0x0C Charger 92,50 µs 9:52 h 52,50 µs Charging Current (0x14) E8 03
35 0x0C Charger 92,50 µs 9:52 h 52,50 µs Charging Voltage (0x15) FF FF
36 0x0C Charger 92,50 µs 9:52 h 52,50 µs Alarm Warning (0x16) 00 80
37 0x29 ADM 1021 A 380,00 µs 9:52 h 6,25 µs Local Temp. Value (0x00) 14
38 0x29 ADM 1021 A 380,00 µs 9:52 h 10,00 µs Remote Temp. Value (0x01) 16
Workspace Find
Time View Message View Live View Statistic View
Dummy [0xEA4A] I2C I2C Vcc: 5,00V Messages: 110

Figure 5.14: Message View: Register Mode

5.5.3 Register Mode

If IRD or plug-in files have been registered, the register mode offers a high-level view of the transferred data. The user can see, which data gets read or written from or to a specific register.

Figure 5.14 shows the register mode. The visualization is based on the same data as the raw mode shown in figure 5.13. So e.g. the first I2C message has been converted from transmitting of ”0x00” and receiving of ”0x00 0x00 0xCD” into reading ”0x00 0x00” from the ”Manufacturer Access” register.

The columns have got the following meaning:

No See section 5.5.2.

Status The two status icons have got the following meaning:

1st Icon Shows whether the I2C messages has been transferred successfully:

no error
I2C address has not been acknowledged
one of the data bytes has not been acknowledged

2nd Icon Shows information about the SMBus feature ”Packet Error Checking” (PEC):

no PEC checksum available
PEC checksum available and ok
PEC checksum available and wrong

Addr See section 5.5.2.

Msg. Time See section 5.5.2.

Abs. Time See section 5.5.2.

Idle Time See section 5.5.2.

Dir The data can be read from the register and written to it. This icon specifies the direction:

data is written to the register
data is read from the register

Register Name of the register. The address pointer value of the register is specified in brackets after the name.

Data Data bytes, which have been read from the register or written to it. Using the context menu it is possible to change the numeric format. The default format is hexadecimal.

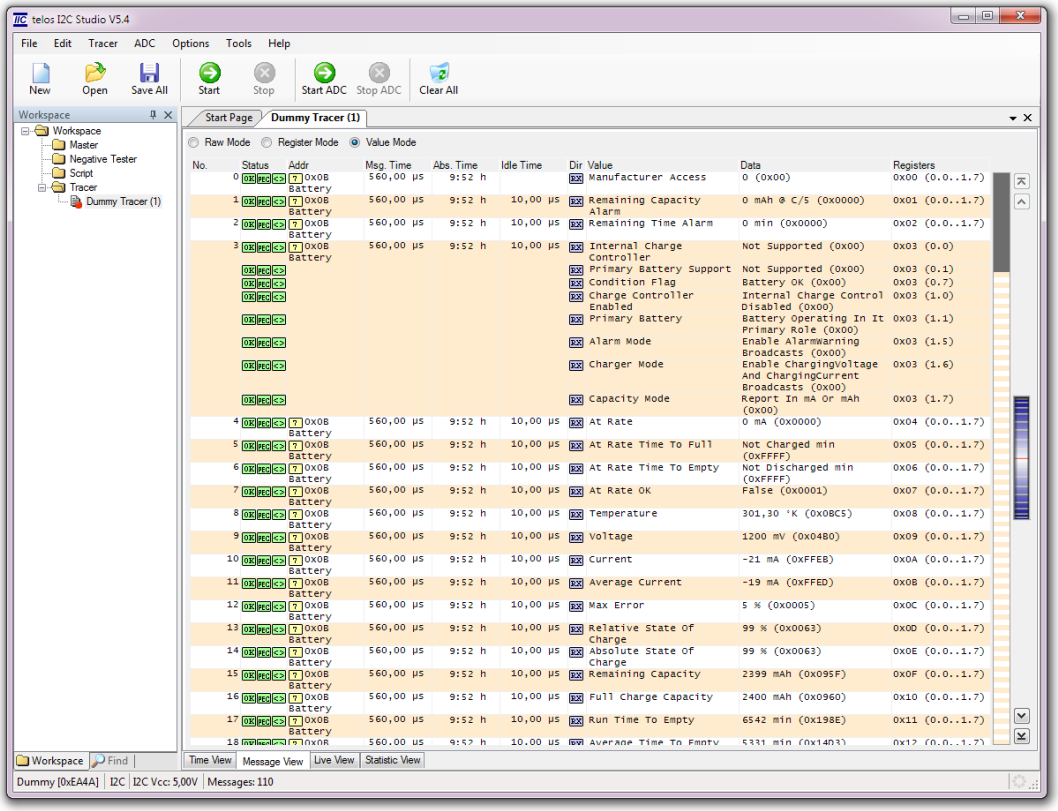

telos I2C Studio V5.4
File Edit Tracer ADC Options Tools Help
New Open Save All Start Stop Start ADC Stop ADC Clear All
Workspace
Workspace
Master
Negative Tester
Script
Tracer
Dummy Tracer (1)
Start Page
Dummy Tracer (1)
Raw Mode
Register Mode
Value Mode
No. Status Addr Msg. Time Abs. Time Idle Time Dir Value Data Registers
0 0x0B Battery 560,00 µs 9:52 h Manufacturer Access 0 (0x00) 0x00 (0.0..1.7)
1 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Capacity Alarm 0 mAh @ C/5 (0x0000) 0x01 (0.0..1.7)
2 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Time Alarm 0 min (0x0000) 0x02 (0.0..1.7)
3 0x0B Battery 560,00 µs 9:52 h 10,00 µs Internal Charge Controller Not Supported (0x00) 0x03 (0.0)
Primary Battery Support Not Supported (0x00) 0x03 (0.1)
Condition Flag Battery OK (0x00) 0x03 (0.7)
Charge Controller Enabled Internal Charge Control Disabled (0x00) 0x03 (1.0)
Primary Battery Battery Operating In It Primary Role (0x00) 0x03 (1.1)
Alarm Mode Enable AlarmWarning Broadcasts (0x00) 0x03 (1.5)
Charger Mode Enable ChargingVoltage And ChargingCurrent Broadcasts (0x00) 0x03 (1.6)
Capacity Mode Report In mA Or mAh (0x00) 0x03 (1.7)
4 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate 0 mA (0x0000) 0x04 (0.0..1.7)
5 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate Time To Full Not Charged min (0xFFFF) 0x05 (0.0..1.7)
6 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate Time To Empty Not Discharged min (0xFFFF) 0x06 (0.0..1.7)
7 0x0B Battery 560,00 µs 9:52 h 10,00 µs At Rate OK False (0x0001) 0x07 (0.0..1.7)
8 0x0B Battery 560,00 µs 9:52 h 10,00 µs Temperature 301,30 °K (0x08C5) 0x08 (0.0..1.7)
9 0x0B Battery 560,00 µs 9:52 h 10,00 µs Voltage 1200 mV (0x04B0) 0x09 (0.0..1.7)
10 0x0B Battery 560,00 µs 9:52 h 10,00 µs Current -21 mA (0xFFEB) 0x0A (0.0..1.7)
11 0x0B Battery 560,00 µs 9:52 h 10,00 µs Average Current -19 mA (0xFFED) 0x0B (0.0..1.7)
12 0x0B Battery 560,00 µs 9:52 h 10,00 µs Max Error 5 % (0x0005) 0x0C (0.0..1.7)
13 0x0B Battery 560,00 µs 9:52 h 10,00 µs Relative State Of Charge 99 % (0x0063) 0x0D (0.0..1.7)
14 0x0B Battery 560,00 µs 9:52 h 10,00 µs Absolute State Of Charge 99 % (0x0063) 0x0E (0.0..1.7)
15 0x0B Battery 560,00 µs 9:52 h 10,00 µs Remaining Capacity 2399 mAh (0x095F) 0x0F (0.0..1.7)
16 0x0B Battery 560,00 µs 9:52 h 10,00 µs Full Charge Capacity 2400 mAh (0x0960) 0x10 (0.0..1.7)
17 0x0B Battery 560,00 µs 9:52 h 10,00 µs Run Time To Empty 6542 min (0x198E) 0x11 (0.0..1.7)
Workspace Find
Time View Message View Live View Statistic View
Dummy [0xEA4A] I2C I2C Vcc: 5,00V Messages: 110

Figure 5.15: Message View: Value Mode

5.5.4 Value Mode

Like the register mode the value mode is based on IRD or plug-in files, too. It offers a visualization of the messages one level higher than the visualization offered by the register mode. In most cases one register does not contain exactly one value. E.g. a 8-bit register can contain eight status bits or a 32-bit value is stored in four 8-bit registers.

This mode converts the transferred raw data to their meaning on the value level. Figure 5.15 shows the message view in the value mode. Lets take message three as an example. The I2C message with the raw data "0x03" and "0x00 0x00 0xF7" has been converted to several values. This transfer reads e.g. "Not Supported" from the "Internal Charge Controller" value of the IC.

The columns of the message view have got the following meaning in this mode:

No See section 5.5.2.

Status The two status icons have got the following meaning:

1st Icon Shows whether the I2C messages has been transferred successfully:

OK	no error
00	I2C address has not been acknowledged
A	one of the data bytes has not been acknowledged

2nd Icon Shows information about the SMBus feature "Packet Error Checking" (PEC):

	no PEC checksum available
PEC	PEC checksum available and ok
A	PEC checksum available and wrong

3rd Icon Shows whether the data is within the allowed range or not:

<>	data is within the range
<>	data is out-of-range / invalid

Addr See section 5.5.2.

Msg. Time See section 5.5.2.

Abs. Time See section 5.5.2.

Idle Time See section 5.5.2.

Dir The data can be read from the register and written to it. This icon specifies the direction:

TX	data is written to the value
RX	data is read from the value

Value Name of the value.

Data Data, which has been read from the value or written to it.

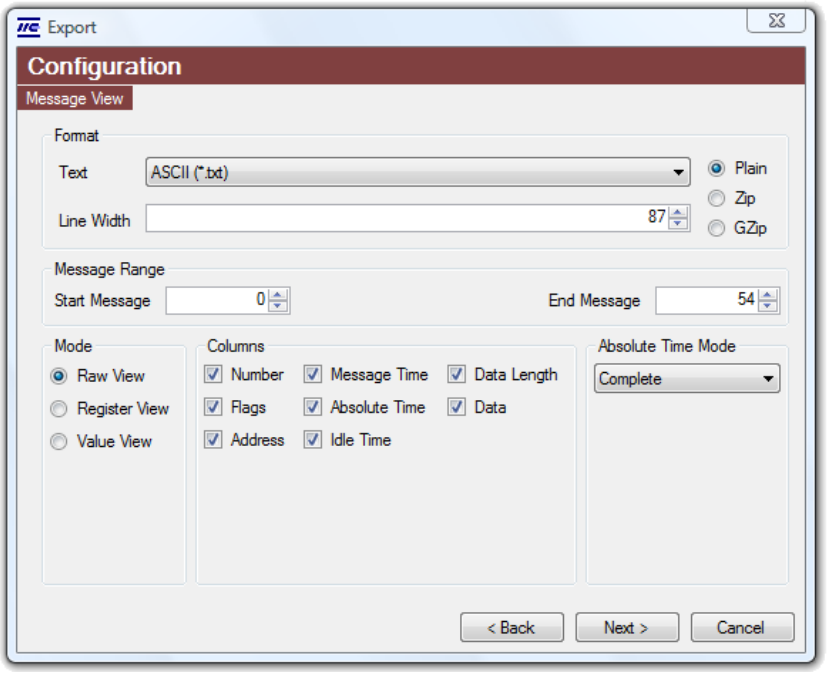


Figure 5.16: Message View: Export

Registers Gives detailed information about the registers, which are used to store the data of this value. The register are specified in the format ”0xaa (b.c) or ”0xaa (b.c..d.e)”. ”aa” is the address pointer of the register, ”b” and ”d” are the n-th byte of the register and ”c” and ”e” are the n-th bit of this byte. So e.g. ”0x10 (0.0..0.7)” stands for the first byte of register ”0x10”.

5.5.5 Export / Copy to Clipboard

The traced I2C data can be exported to an external file or to the clipboard using the visualization of the message view. The exporting to a file gets configured using a wizard, which can be started via the File|Trace Data|Export menu item. After selecting ”Message View” on the first page of the wizard, the wizard should look like that one, which is shown in figure 5.16.

To export the data to the system’s clipboard, simply select Copy to Clipboard in the context menu of the message view.

The option page of the export wizard offers the following options:

Text The user can select between three different formats:

ASCII (*.txt) The data gets exported to a formatted ASCII file. Using the ”Line Width” parameter it is possible to configure the width of the created table. This format is humanly readable.

Character Separated Values (*.csv) The CSV format can be imported into a lot of programs like e.g. Microsoft Excel or OpenOffice. Furthermore it is very simply to write own parsers for this format. It conforms to RFC 4180. With the ”Seperator”

switch the user can select, whether the cells in the file should be seperated with commas or semicolons.

HyperText Markup Language (*.html) Creates a humanly readable HTML file, which can be viewed with most standard WWW browsers like e.g. Mozilla or the Internet Explorer. It conforms to XHTML 1.0.

Compression If the data is exported to a file, it is possible to compress the data:

Plain Do not use any compression.

Zip Use the ZIP format, which is supported by most compression tools on MS Windows like e.g. WinZip or 7-Zip.

GZip Use the gzip format, which is very common in the Linux world, but also supported by most tools on MS Windows.

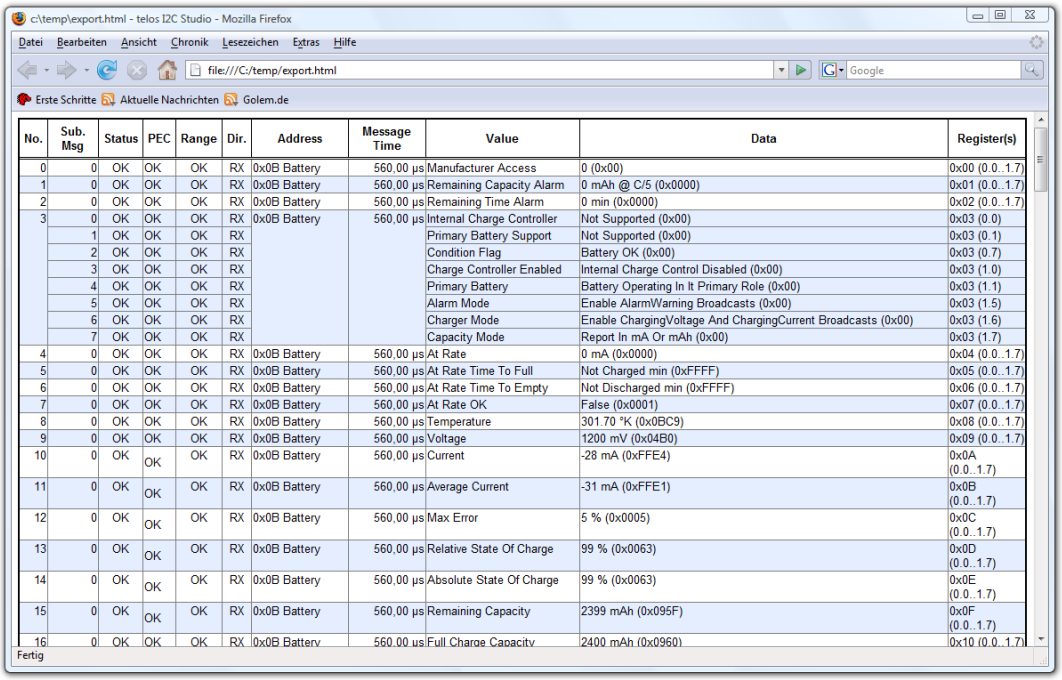
Message Range Specifies the messages, which should be exported.

Mode Like in the GUI there are three modes, which can be chosen.

Columns Defines the columns, which should be exported. The available columns depend on the selected mode.

Absolute Time Mode Configures the format of the absolute time column. The user can choose between three different formats. The formats ”Short” and ”Complete” refer to the real-time clock of the PC. In the format ”Relative” the time is relative to the time, when the I2C tracer has been enabled the first time.

After the configuration has been completed the Next > button opens the third page of the wizard. This page allows the user to specify a file name. The Finish button closes the wizard and starts the exporting of the data. Figure 5.17 shows some data, which has been exported to the HTML format.



c:\temp\export.html - telos I2C Studio - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

file:///C:/temp/export.html

Erste Schritte Aktuelle Nachrichten Golem.de

No.	Sub. Msg	Status	PEC	Range	Dir.	Address	Message Time	Value	Data	Register(s)
0	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Manufacturer Access	0 (0x00)	0x00 (0.0..1.7)
1	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Remaining Capacity Alarm	0 mAh @ C/5 (0x0000)	0x01 (0.0..1.7)
2	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Remaining Time Alarm	0 min (0x0000)	0x02 (0.0..1.7)
3	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Internal Charge Controller	Not Supported (0x00)	0x03 (0.0)
	1	OK	OK	OK	RX			Primary Battery Support	Not Supported (0x00)	0x03 (0.1)
	2	OK	OK	OK	RX			Condition Flag	Battery OK (0x00)	0x03 (0.7)
	3	OK	OK	OK	RX			Charge Controller Enabled	Internal Charge Control Disabled (0x00)	0x03 (1.0)
	4	OK	OK	OK	RX			Primary Battery	Battery Operating In It Primary Role (0x00)	0x03 (1.1)
	5	OK	OK	OK	RX			Alarm Mode	Enable AlarmWarning Broadcasts (0x00)	0x03 (1.5)
	6	OK	OK	OK	RX			Charger Mode	Enable ChargingVoltage And ChargingCurrent Broadcasts (0x00)	0x03 (1.6)
	7	OK	OK	OK	RX			Capacity Mode	Report In mA Or mAh (0x00)	0x03 (1.7)
4	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	At Rate	0 mA (0x0000)	0x04 (0.0..1.7)
5	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	At Rate Time To Full	Not Charged min (0xFFFF)	0x05 (0.0..1.7)
6	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	At Rate Time To Empty	Not Discharged min (0xFFFF)	0x06 (0.0..1.7)
7	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	At Rate OK	False (0x0001)	0x07 (0.0..1.7)
8	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Temperature	301.70 °K (0x0BC9)	0x08 (0.0..1.7)
9	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Voltage	1200 mV (0x04B0)	0x09 (0.0..1.7)
10	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Current	-28 mA (0xFFE4)	0x0A (0.0..1.7)
11	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Average Current	-31 mA (0xFFE1)	0x0B (0.0..1.7)
12	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Max Error	5 % (0x0005)	0x0C (0.0..1.7)
13	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Relative State Of Charge	99 % (0x0063)	0x0D (0.0..1.7)
14	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Absolute State Of Charge	99 % (0x0063)	0x0E (0.0..1.7)
15	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Remaining Capacity	2399 mAh (0x095F)	0x0F (0.0..1.7)
16	0	OK	OK	OK	RX	0x0B Battery	560,00 µs	Full Charge Capacity	2400 mAh (0x0960)	0x10 (0.0..1.7)

Fertig

Figure 5.17: Message View: Exported Data (HTML)

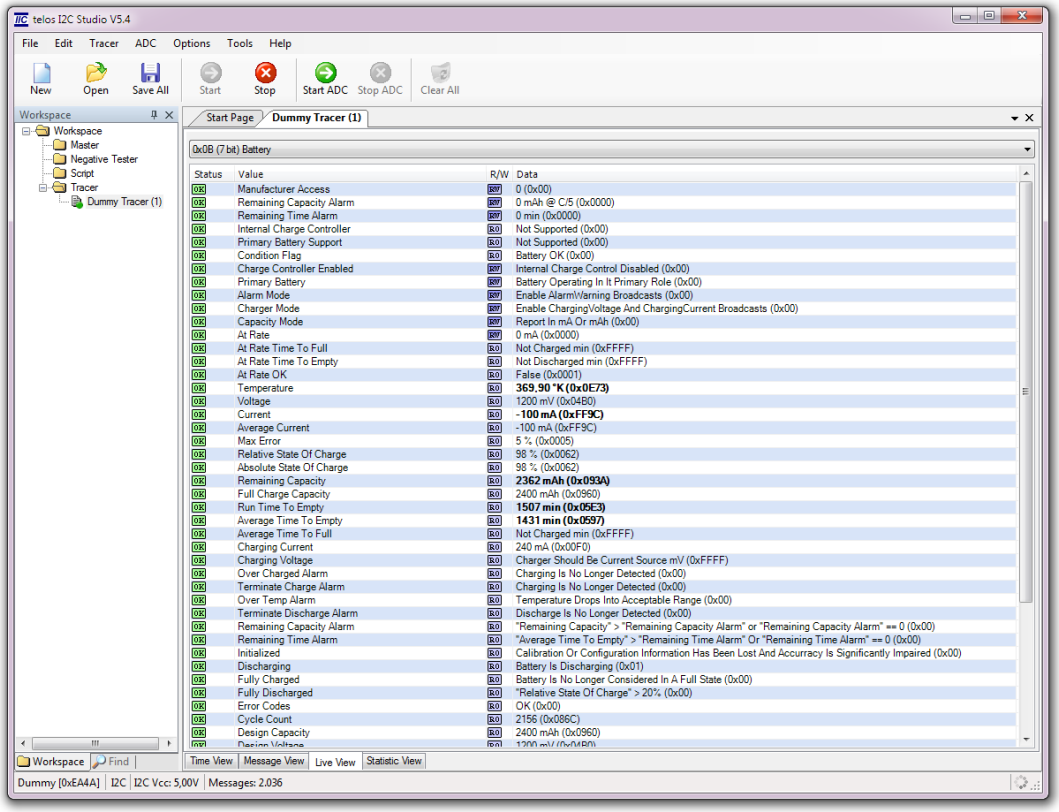

telos I2C Studio V5.4
File Edit Tracer ADC Options Tools Help
New Open Save All Start Stop Start ADC Stop ADC Clear All
Workspace
Workspace
Master
Negative Tester
Script
Tracer
Dummy Tracer (1)
Start Page
Dummy Tracer (1)
0x0B (7 bit) Battery
Status Value R/W Data
Manufacturer Access 0 (0x00)
Remaining Capacity Alarm 0 mAh @ C/5 (0x0000)
Remaining Time Alarm 0 min (0x0000)
Internal Charge Controller Not Supported (0x00)
Primary Battery Support Not Supported (0x00)
Condition Flag Battery OK (0x00)
Charge Controller Enabled Internal Charge Control Disabled (0x00)
Primary Battery Battery Operating In It Primary Role (0x00)
Alarm Mode Enable Alarm\Warning Broadcasts (0x00)
Charger Mode Enable ChargingVoltage And ChargingCurrent Broadcasts (0x00)
Capacity Mode Report In mA Or mAh (0x00)
At Rate 0 mA (0x0000)
At Rate Time To Full Not Charged min (0xFFFF)
At Rate Time To Empty Not Discharged min (0xFFFF)
At Rate OK False (0x0001)
Temperature 369,90 °K (0x0E73)
Voltage 1200 mV (0x04B0)
Current -100 mA (0xFF9C)
Average Current -100 mA (0xFF9C)
Max Error 5 % (0x0005)
Relative State Of Charge 98 % (0x0062)
Absolute State Of Charge 98 % (0x0062)
Remaining Capacity 2362 mAh (0x093A)
Full Charge Capacity 2400 mAh (0x0960)
Run Time To Empty 1507 min (0x05E3)
Average Time To Empty 1431 min (0x0597)
Average Time To Full Not Charged min (0xFFFF)
Charging Current 240 mA (0x00F0)
Charging Voltage Charger Should Be Current Source mV (0xFFFF)
Over Charged Alarm Charging Is No Longer Detected (0x00)
Terminate Charge Alarm Charging Is No Longer Detected (0x00)
Over Temp Alarm Temperature Drops Into Acceptable Range (0x00)
Terminate Discharge Alarm Discharge Is No Longer Detected (0x00)
Remaining Capacity Alarm "Remaining Capacity" > "Remaining Capacity Alarm" or "Remaining Capacity Alarm" == 0 (0x00)
Remaining Time Alarm "Average Time To Empty" > "Remaining Time Alarm" Or "Remaining Time Alarm" == 0 (0x00)
Initialized Calibration Or Configuration Information Has Been Lost And Accurracy Is Significantly Impaired (0x00)
Discharging Battery Is Discharging (0x01)
Fully Charged Battery Is No Longer Considered In A Full State (0x00)
Fully Discharged "Relative State Of Charge" > 20% (0x00)
Error Codes OK (0x00)
Cycle Count 2156 (0x086C)
Design Capacity 2400 mAh (0x0960)
Workspace Find
Time View Message View Live View Statistic View
Dummy [0xEA4A] I2C I2C Vcc: 5,00V Messages: 2.036

Figure 5.18: Live View

5.6 Live View

The live view shows the user the content, which is currently stored in the values of the devices on the monitored I2C bus.

This view is based on the information, which is stored in IRD files. So it can be used only with I2C devices, for which appropriate IRD files are available and have been registered.

When the I2C tracer is running, this view permanently parses the incoming I2C data to monitor any changes of the displayed values. Due to this fact the live view can display only modifications, which have been seen on the I2C bus. If an I2C device changes e.g. a status value internally and this value is never read by an external I2C master, the view will not update its view of the value to the new content. This limitation is equalized by the fact that the view does not need to send any data actively to the I2C bus to read out the devices.

In the upper part of the live view there is a combo box, which can be used to select whose values should be displayed. After one I2C device has been selected, its current values get displayed in the table. The columns of the table have got the following meaning:

Status Shows the status of the value:

OK	The current content is valid.
<>	The current content is out-of-range.

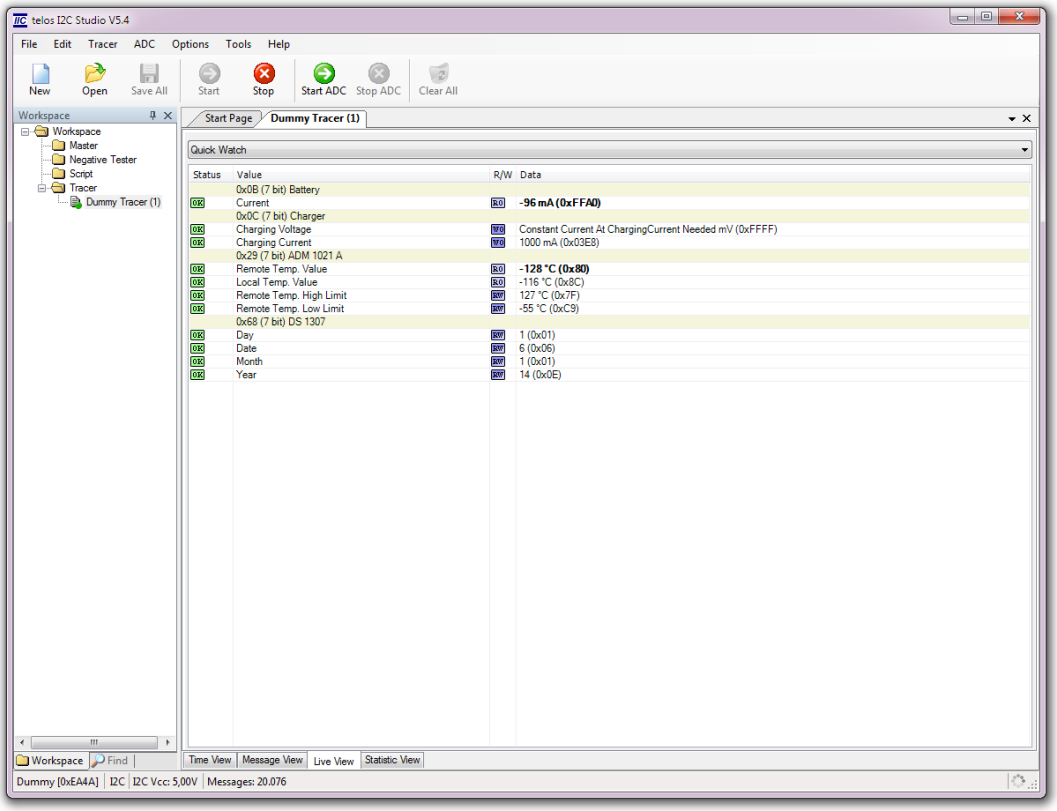

telos I2C Studio V5.4
File Edit Tracer ADC Options Tools Help
New Open Save All Start Stop Start ADC Stop ADC Clear All
Workspace
Workspace
Master
Negative Tester
Script
Tracer
Dummy Tracer (1)
Start Page
Dummy Tracer (1)
Quick Watch
Status Value R/W Data
0x0B (7 bit) Battery
OK Current RO -96 mA (0xFFA0)
0x0C (7 bit) Charger
OK Charging Voltage WO Constant Current At ChargingCurrent Needed mV (0xFFFF)
OK Charging Current WO 1000 mA (0x03E8)
0x29 (7 bit) ADM 1021 A
OK Remote Temp. Value RO -128 °C (0x80)
OK Local Temp. Value RO -116 °C (0x8C)
OK Remote Temp. High Limit RW 127 °C (0x7F)
OK Remote Temp. Low Limit RW -55 °C (0xC9)
0x68 (7 bit) DS 1307
OK Day RW 1 (0x01)
OK Date RW 6 (0x06)
OK Month RW 1 (0x01)
OK Year RW 14 (0x0E)
Workspace Find
Time View Message View Live View Statistic View
Dummy [0xEA4A] I2C I2C Vcc: 5,00V Messages: 20.076

Figure 5.19: Live View: Quick Watch

Value Name of the value.

R/W Some values are write-only or read-only. This column specifies the allowed directions.

RO	read-only
WO	write-only
RW	read/write

Data Contains the current content of the value. If it is not known, the table shows ”Unknown Value”.

The lines in the table, whose data have changed a minute ago, are highlighted with a bold face font.

When working with devices, which contain not only a few values but a large quantity of values, it can be difficult to find the needed values in the tables. To make life easier I2C Studio offers a feature called ”quick watch”. The idea is to add the values of interest into a separate table.

To add a value to the quick watch simply select the appropriate row in the table and use the context menu item Add To Quick Watch. After the values have been added, the user can change to the quick watch using the same combo box, which is normally used to select the displayed device. Figure 5.19 shows a quick watch to which some values have been added.

Values can be removed from the quick watch by the context menu item Remove From Quick Watch, which is available in the quick watch window only.

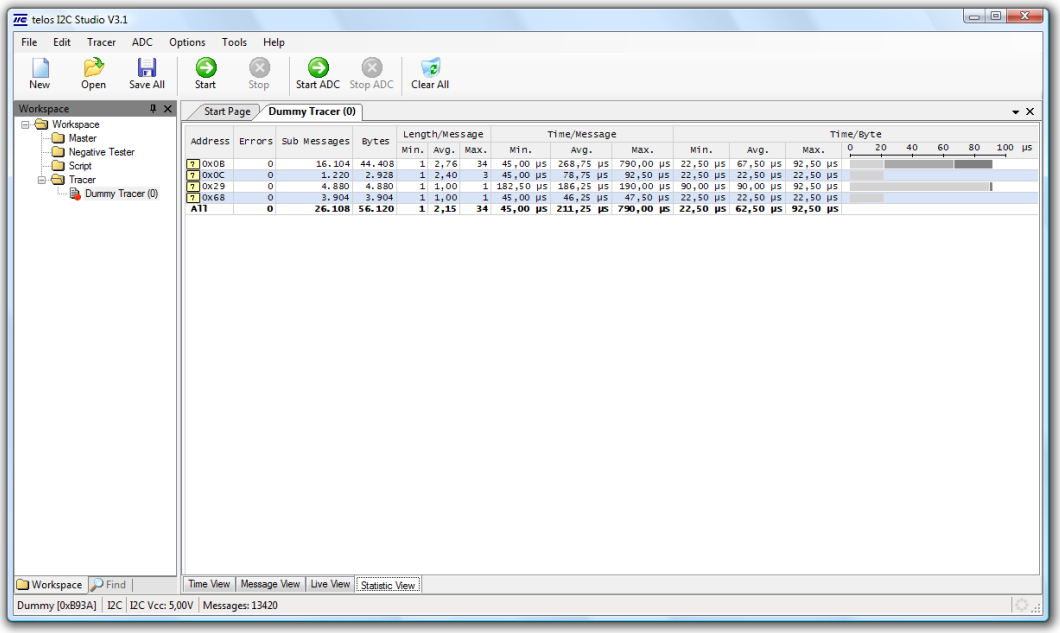

telos I2C Studio V3.1
File Edit Tracer ADC Options Tools Help
New Open Save All Start Stop Start ADC Stop ADC Clear All
Workspace
Workspace
Master
Negative Tester
Script
Tracer
Dummy Tracer (0)
Start Page
Dummy Tracer (0)
Address Errors Sub Messages Bytes Length/Message Time/Message Time/Byte
Min. Avg. Max.
0x0B 0 16.104 44.408 1 2,76 34 45,00 µs 268,75 µs 790,00 µs 22,50 µs 67,50 µs 92,50 µs
0x0C 0 1.220 2.928 1 2,40 3 45,00 µs 78,75 µs 92,50 µs 22,50 µs 22,50 µs 22,50 µs
0x29 0 4.880 4.880 1 1,00 1 182,50 µs 186,25 µs 190,00 µs 90,00 µs 90,00 µs 92,50 µs
0x68 0 3.904 3.904 1 1,00 1 45,00 µs 46,25 µs 47,50 µs 22,50 µs 22,50 µs 22,50 µs
All 0 26.108 56.120 1 2,15 34 45,00 µs 211,25 µs 790,00 µs 22,50 µs 62,50 µs 92,50 µs
Workspace Find
Time View Message View Live View Statistic View
Dummy [0xB93A] I2C I2C Vcc: 5,00V Messages: 13420

Figure 5.20: Statistic View

5.7 Statistic View

The statistic view displays a statistical analysis of the submessages, which have been traced on the I2C bus. Using this view the developer gets an overview e.g. which I2C addresses have been accessed how many times or how long is the average submessage to the I2C slave at address 0x50.

Figure 5.20 shows the statistic view. The table of the view contains a row for each I2C address, to which submessages have been sent. The last row, which is called ”All”, contains the sum of all rows.

Note: The statistic is based on I2C submessages. If e.g. an I2C message consists of two submessages, these submessages are analysed separately.

5.7.1 Statistic

The columns of the table contain the following statistics:

Address I2C address of the slave.

The icons can have got the following meaning:

7	7-bit I2C address
8	7-bit I2C address (8-bit representation)
10	10-bit I2C address

In the case of transmission errors, it is possible that the lower part of a 10-bit address is missing. In such cases rows with I2C addresses like ”0x0XX” are shown in the table.

Beside the normal I2C addresses the following values are possible in this column:

Prot. Error An I2C protocol error can occur e.g. when a STOP condition is not sent after the 9th data bit but after the 2nd.

SMB Pwr Off The power of a SMBus has been switched off.

SMB Pwr On The power of a SMBus has been switched on.

N/A The I2C address is missing, because e.g. the tracing starts in the middle of an I2C message so that the START condition has not been recorded.

The interpretation of the I2C address can be switched from the default 7-bit mode to the 8-bit mode, which is used by some data-sheets. This is done using the context menu switch Hex Radix. Moreover the user can select a decimal instead of the default hexadecimal interpretation using the 7 Bit Address item.

Errors Contains the number of I2C submessages, which has been transmitted with errors. If the I2C address or one of the data bytes has not been acknowledged, this is counted as error.

Sub Messages Number of submessages, which have been sent to this I2C address.

Bytes Sum of the data bytes, which have been exchanged with the I2C slave at this address. This value does not include the bytes, which are used to transmit the I2C address on the I2C bus.

Length/Message Number of data bytes contained in the submessages exchanged with this slave. The minimum, average, and maximum values are shown. This value does not include the bytes, which are used to transmit the I2C address on the I2C bus.

Time/Message Length of the submessages as time. The measured time starts with the START condition and ends with the STOP condition of the submessage. If the submessage does not contain a STOP condition, because it is part of a combined transfer, the time ends with the acknowledge bit of the last data byte. The minimum, average, and maximum values are shown.

Time/Bytes Length of the bytes in the submessages as time. This value takes into account not only the data bytes but also the bytes, which have been used to transmit the I2C address.

With the context menu of the statistic view, which is shown in figure 5.21, the user can disable or enable the various columns of the table. Furthermore one value can be chosen, whose result should be visualized with a bargraph. This bargraph displays up to three values with different grey levels: the minimum, average, and maximum.

5.7.2 Export / Copy to Clipboard

The statistic can be exported to a file or to the system's clipboard. The exporting to a file can be started via the menu item File|Trace Data|Export whereas the exporting to the clipboard is started using the context menu item Copy to Clipboard.

In both cases a wizard gets started. Figure 5.22 shows the options, which can be configured:

Text Three different formats are available:

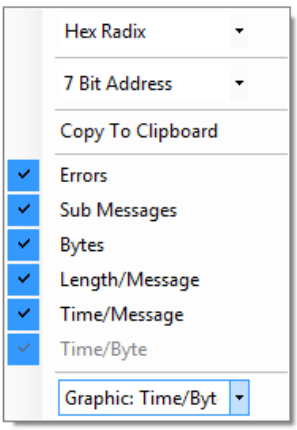


Figure 5.21: Statistic View: Context Menu

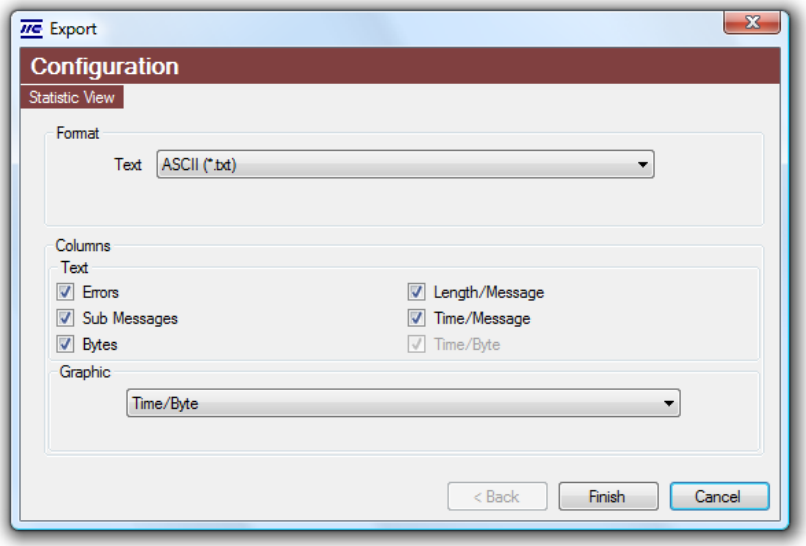
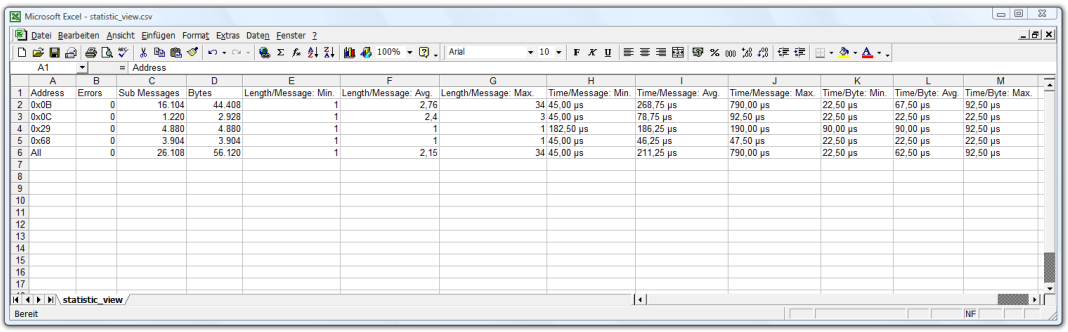


Figure 5.22: Statistic View: Export



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Address	Errors	Sub Messages	Bytes	Length/Message: Min.	Length/Message: Avg.	Length/Message: Max.	Time/Message: Min.	Time/Message: Avg.	Time/Message: Max.	Time/Byte: Min.	Time/Byte: Avg.	Time/Byte: Max.
2	0x0B	0	16.104	44.408	1	2,76	34	45,00 µs	268,75 µs	790,00 µs	22,50 µs	67,50 µs	92,50 µs
3	0x0C	0	1.220	2.928	1	2,4	3	45,00 µs	78,75 µs	92,50 µs	22,50 µs	22,50 µs	22,50 µs
4	0x29	0	4.880	4.880	1	1	1	182,50 µs	186,25 µs	190,00 µs	90,00 µs	90,00 µs	92,50 µs
5	0x68	0	3.904	3.904	1	1	1	45,00 µs	46,25 µs	47,50 µs	22,50 µs	22,50 µs	22,50 µs
6	All	0	26.108	56.120	1	2,15	34	45,00 µs	211,25 µs	790,00 µs	22,50 µs	62,50 µs	92,50 µs

Figure 5.23: Statistics Imported Into MS Excel

ASCII (*.txt) The data gets exported to a formatted ASCII file. This format is humanly readable.

Character Separated Values (*.csv) The CSV format can be imported into a lot of programs like e.g. Microsoft Excel or OpenOffice. Furthermore it is very simply to write ones own parsers for this format. It conforms to RFC 4180. With the "Separator" switch the user can select, whether the cells in the file should be seperated with commas or semicolons.

HyperText Markup Language (*.html) Creates a humanly readable HTML file, which can be viewed with most standard WWW browsers like e.g. Mozilla or the Internet Explorer. It conforms to XHTML 1.0.

Columns Defines the statistics to be exported.

Graphic In some formats it is possible to visualize one of the values with an additional bargraph. This option defines, which value should be exported with a bargraph.

Figure 5.23 shows a CSV file loaded into MS Excel, which has been created by I2C Studio.

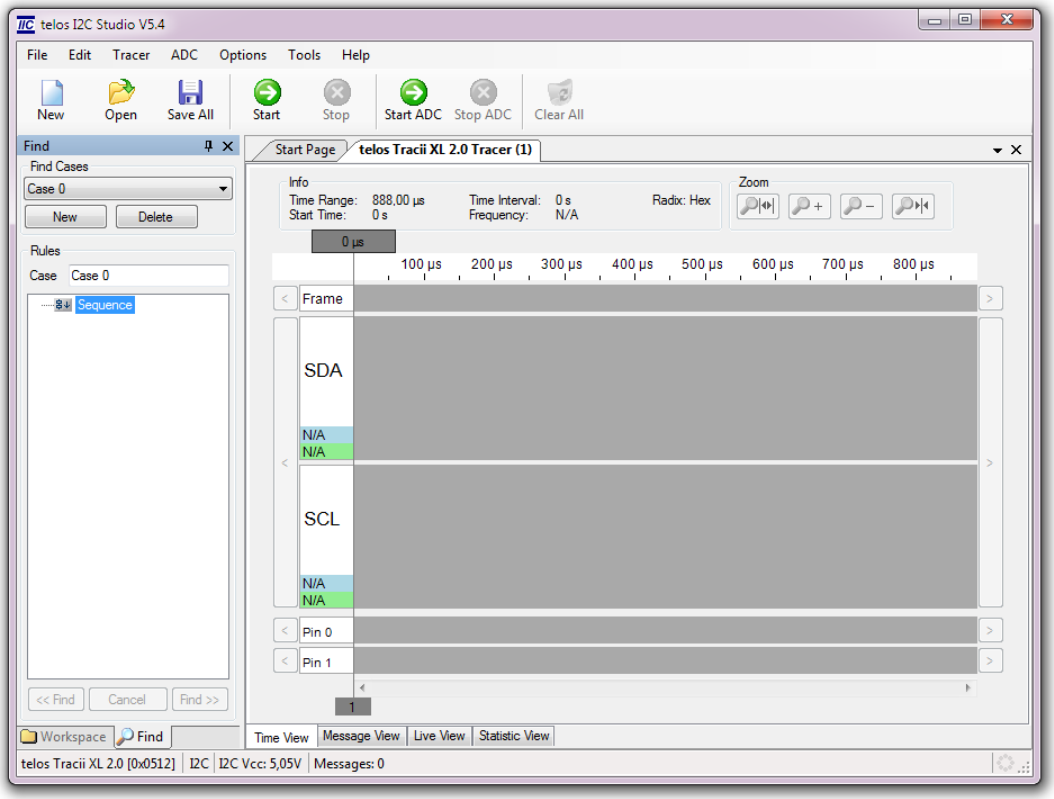


Figure 5.24: Find Window

5.8 Find

5.8.1 Find Window

I2C Studio offers only an I2C address filter to limit the messages, which get recorded. This can result into a very large number of messages, if there is heavy load on the examined I2C bus. So in the worst case up to 20,000 messages/sec are recorded on an I2C bus, whose master work with 400 kHz.

After recording data on such a bus e.g. for one hour, it is nearly impossible to find the interesting messages by hand. That is why I2C Studio comes with a find window allowing to define very complex search criterions.

Figure 5.24 shows the empty find window. Within this window the user can specify the rules, which should be used to search for the interesting data.

Under "Find Cases" the user can store several collections of find rules, so that he can switch between different rule sets without deleting and entering them again and again. A new collection is started by clicking onto the New button. A no longer needed collection can be removed by clicking onto the Delete button.

The "Rules" area of the window allows to specify the search criterions for one rule set. The rules are organized in a tree. The icons in the tree have got the following meaning:

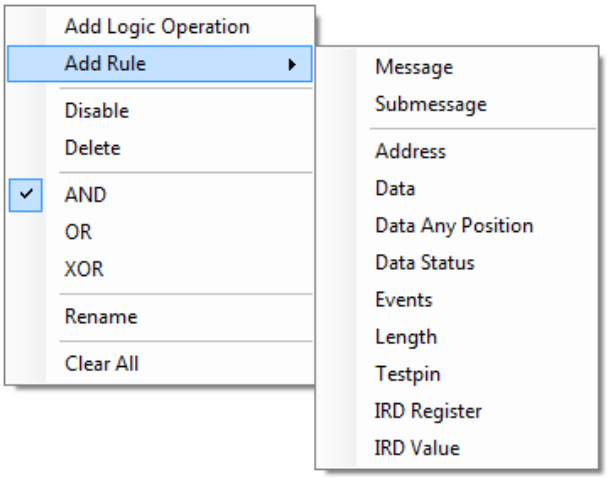


Figure 5.25: Find Window: Context Menu

S↓	Sequence
AND	Logical AND
OR	Logical OR
XOR	Logical XOR
==	Rule

After the user has entered the rules, the search can be started by clicking onto the Find >> button to search forward and onto the << Find button to search backwards. If the developer has recorded a lot of I2C messages, the search process can take some time. There is a Cancel button to stop a running search process. The current progress of the search gets visualized in a progress bar at the lower part of the window.

5.8.2 Rules

There are three groups of items in the tree of the find window: the sequence, logical operations, and the rules.

The criterions within the sequence must be fulfilled sequentially. So after the first criterion has been matched, the next one must be matched starting from the point where the first has been matched, and so on. The sequence forms the top level of the tree.

Into the sequence the user can add logical operations. All rules and logical operations contained into a logical operation get combined logical.

A rule defines the real search criterion. There are different types of rules, e.g. it is possible to define an address rule.

The tree is modified completely with its context menu, see figure 5.25. The context menu knows the following items:

Add Sequence Adds an empty sequence.

Add Logic Operation Adds a logic operation. The results of all children under this operation get combined logically. By default "AND" is used for this combination.

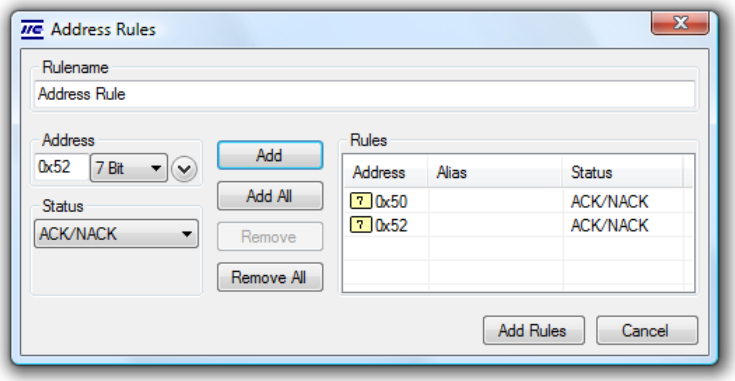


Figure 5.26: Find Window: Address Rules

Add Rule Adds a rule.

Disable Disables the item. Its result does not contribute to the search.

Delete Deletes the item permanently.

AND Chooses the kind of used logical operation: AND.

OR Chooses the kind of used logical operation: OR.

XOR Chooses the kind of used logical operation: XOR.

Rename Renames the item.

Clear Removes everything from the tree.

Address Rules

This type of rule can be used to search for I2C messages to slaves with specific I2C addresses. Besides the I2C addresses the user can specify the status of the I2C addresses:

Address I2C address to be searched for.

Status Status of the I2C address. The following values are possible:

ACK I2C address has been acknowledged.

NACK I2C address has not been acknowledged.

ACK/NACK I2C address has been or has been not acknowledged.

No confirmation The ACK/NACK bit is missing.

The Add All button inserts all possible 7-bit and 10-bit I2C addresses with status "ACK/NACK" to the list. With the Remove All button all rules can be removed from the table.

The rules shown in figure 5.26 match all I2C messages, which have been sent to the slaves with the I2C addresses 0x50 and 0x52 and whose ACK/NACK bit is available.

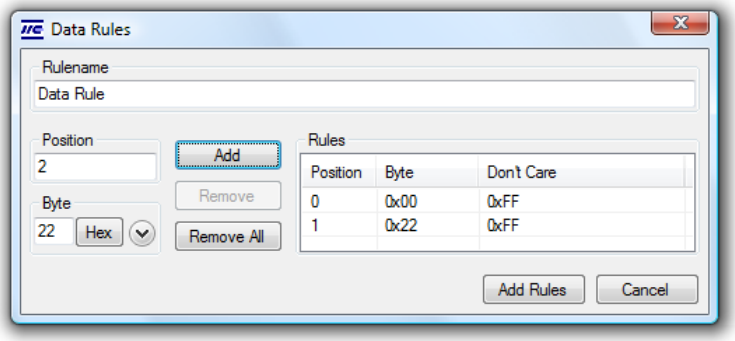


Figure 5.27: Find Window: Data Rules

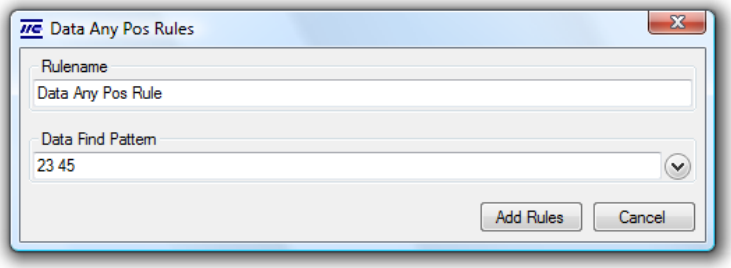


Figure 5.28: Find Window: Data Any Pos Rules

Data Rules

The data rules match bits or complete bytes of the data transferred with I2C messages.

To add e.g. a search for I2C messages, which contain 0x10 as the first data byte, enter "0" to the "Position" and "0x10" to the "Byte" field. Afterwards press Add to add the value to the "Rules" table. To add search rules for bits click on the Down button in the "Byte" field.

The rules, which can be seen in figure 5.27, match all I2C messages, which have "0x00" as first and "0x22" as second data byte.

Note: The bytes, which are used to transfer the I2C addresses on the I2C bus, are not counted to the data bytes.

Data Any Pos Rules

This type of rule is similar to the data rule. It searches for a number of consecutive data bytes. In contrast to the data rule the user cannot specify the position within an I2C message, where the given data byte is expected.

The example in 5.28 searches of the sequence "0x12, 0x34".

Data Status Rules

This rule can be used to find I2C messages, where the I2C slave has acknowledged or not acknowledged the data bytes. If this rule is set to NACK, all I2C messages will be found where at least one data byte has not been acknowledged.

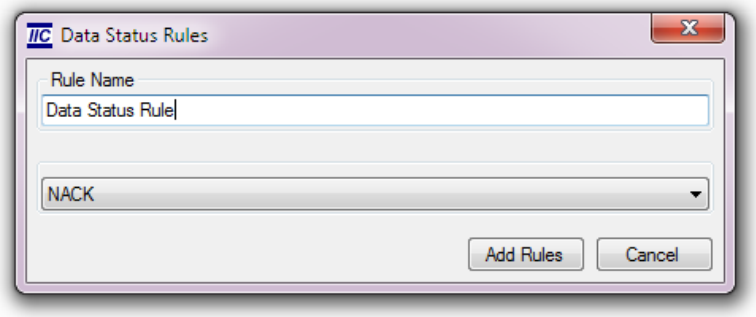


Figure 5.29: Find Window: Data Status Rules

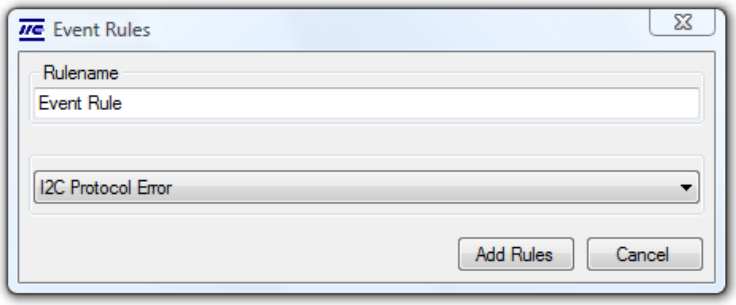


Figure 5.30: Find Window: Event Rules

The example in figure 5.29 searches for NACK.

Event Rules

I2C Studio displays messages in the tracer window, which are in fact no real I2C messages. Instead they are events. This rule type can be used to search for these events.

One of the following events can be chosen:

I2C Protocol Error A violation of the I2C protocol has been detected, e.g. after five data bits a STOP condition has been sent.

SMBus Power Down The power of a SMBus has been switched off.

SMBus Power Up The power of a SMBus has been switched on.

Tracer Restarted The tracer has been restarted.

The example in figure 5.30 searches for an I2C protocol error.

Length Rules

To search for I2C messages, whose submessages contain a specific number of data bytes, this rule can be used.

The rule in figure 5.31 matches I2C messages, whose submessages consist of at least 12 and up to 14 data bytes.

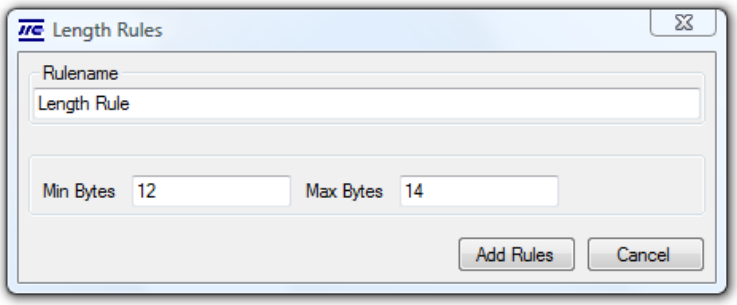


Figure 5.31: Find Window: Length Rules

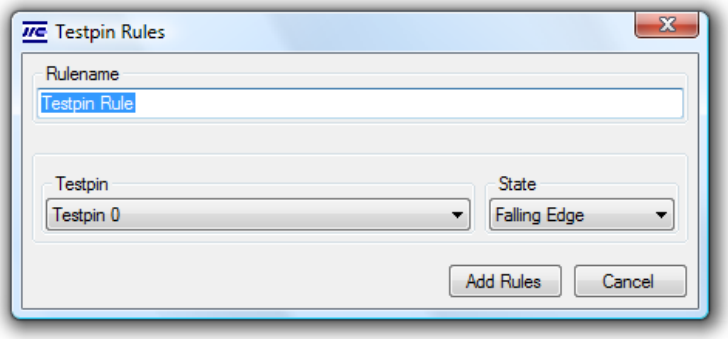


Figure 5.32: Find Window: Testpin Rules

Testpin Rules

The find rules cannot only be used to search within I2C messages, but also to search for specific states of the input testpins or specific transitions on the input testpins.

Note: There are some limitations in the precision of this rule. It will always match the I2C messages, which has been sent in parallel or before the searched state of the testpin.

The rule shown in figure 5.32 will match, when a falling edge is detected on the first input testpin line.

Submessage Rules

This rule consists of a combination of some of the previous mentioned rule types and some additional rule types. This rule matches I2C messages, which contain an I2C submessage fulfilling the search criterions specified by this rule.

There are several sub criterions, which can be enabled or disabled using check boxes:

Length Length of the data, see section 5.8.2.

Address I2C address of the slave to which this submessage has been sent, see section 5.8.2.

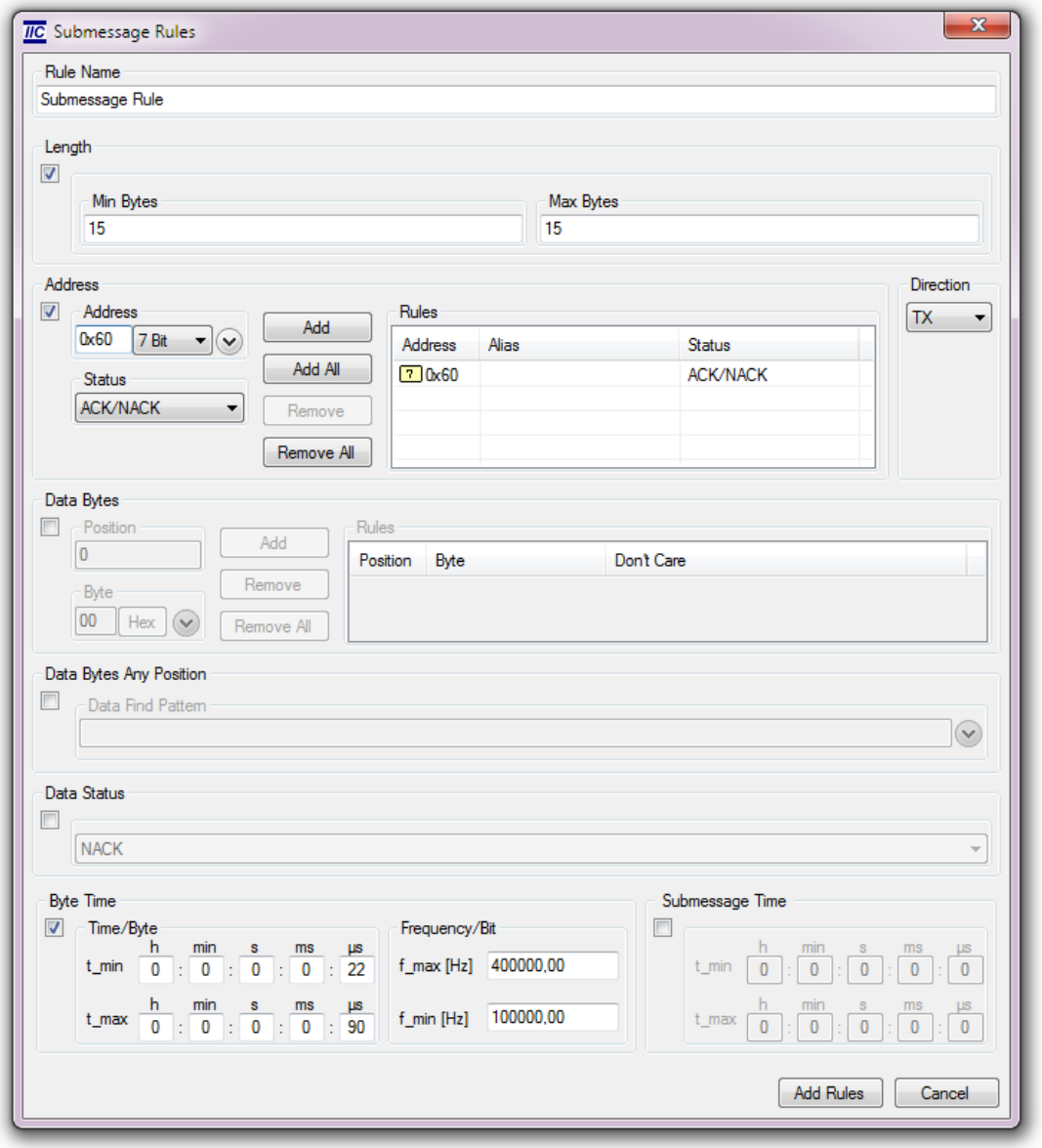


Figure 5.33: Find Window: Submessage Rules

Direction If this option is not enabled, the rule will match submessages independent of their transfer direction. The user can enable this option and choose ”TX” or ”RX” to match only master transmitter or master receiver transfers.

Data Bytes Matches data bytes at specific positions, see section 5.8.2.

Data Bytes Any Pos Matches a sequence of data bytes, see section 5.8.2.

Data Status Matches submessages with the specified data status 5.8.2.

Byte Time Only I2C submessages will match, whose transmission of each data byte has lasted a time within the given time range. The time range can be specific also as frequency. This frequency refers to one bit. A data byte on the I2C bus consists of eight data bits and the acknowledge bit. This criterion can be used to find I2C submessages, which have been sent with a specific clock rate (SCL) or to find e.g. bus stretching.

Submessage Time The transmission time of the I2C submessage must be within the given time range to match the rule.

The example matches all I2C messages, which contain an master transmitter submessage. This submessage has to be sent to the slave with the address 0x60. It has to contain 15 data bytes and has to be sent with a speed between 100 kHz and 400 kHz.

Message Rules

Using this type of rule it is possible to define search criterions on I2C message level. The following sub criterions can be enabled or disabled using check boxes:

Submessages Contains a number of submessage rules as they have been described in section 5.8.2. A new rule can be added to the list with the help of the Add button. To remove one or all rules the Remove and Remove All buttons can be used.

Each rule is connected with a position in the I2C message. In the example shown in figure 5.34 each first submessage of an I2C message is compared with the ”TX Rule” and each second submessage is compared with the ”RX Rule”.

The Change button can be used to change the position of a rule.

Message Time The message time is the time from the first START condition of an I2C message to its STOP condition. An I2C message matches this rule, if its message time is within the specified time range.

Idle Time The idle time is the time from the STOP condition of the previous I2C message to the first START condition of the current I2C message.

Testpins Matches the state of one of the input testpins, see section 5.8.2.

Figure 5.34 shows a rule, which will match I2C messages fulfilling the ”TX Rule” for their first submessage and the ”RX Rule” for the second submessage. Furthermore there must be a falling edge on the first testpin input line.

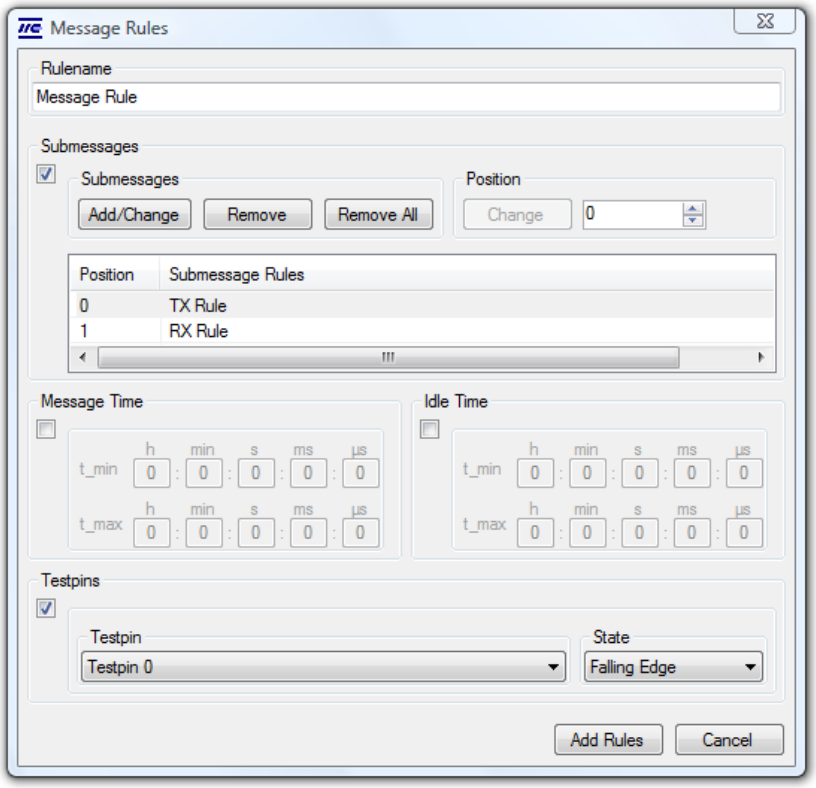


Figure 5.34: Find Window: Message Rules

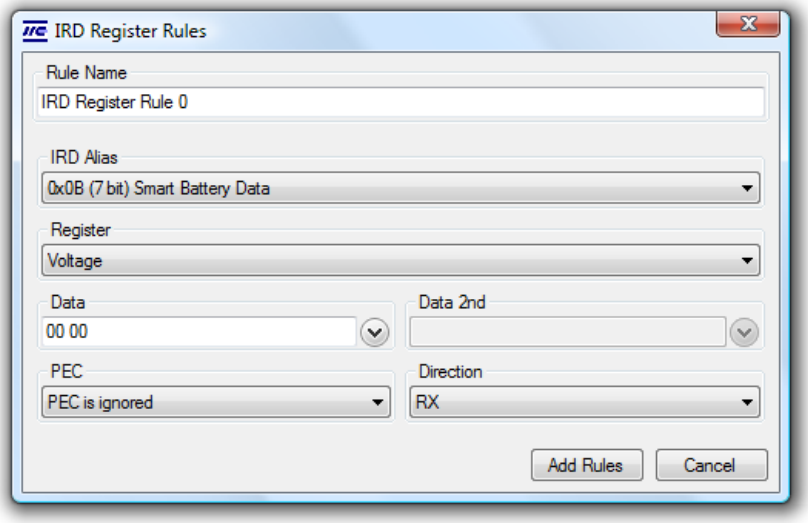


Figure 5.35: Find Window: IRD Register Rules

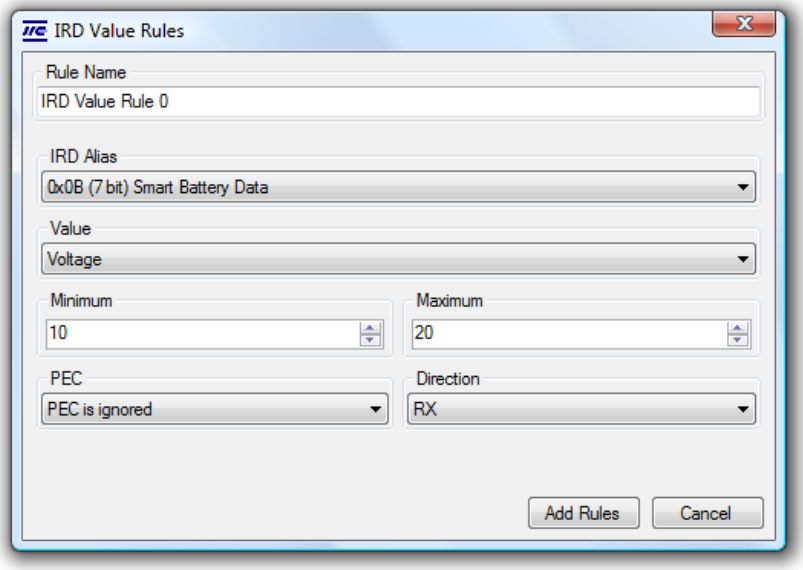


Figure 5.36: Find Window: IRD Value Rules

IRD Register Rules

If an IRD file has been registered, this type of rule can be used to define search criterions on register level. The following sub criterions can be set:

IRD Alias Selects the I2C slave, which has been addressed by the I2C master. Only I2C slaves are shown here, for which IRD files have been registered (see section 5.2.2).

Register Register of the I2C slave.

Data Data, which has been read from the register or which has been written to the register.

Data 2nd Some registers must be written and read within on I2C message. The data for the write transfer must be specified in the ”Data” field and the data for the read transfer must be specified in the ”Data 2nd” field.

PEC Some SMBus slaves support ”packet error checking”. For such slaves this field can be used to search for I2C messages with a specific PEC state.

Direction Specifies the direction of the transfer. If ”TX” is specified only transfers to the register are found. If ”RX” is specified only transfers from the register are found.

The rule shown in figure 5.35 will match, when the ”Voltage” register of the ”Smart Battery Data” device at address ”0x0b” is read and the register contains the value ”00 00”.

IRD Value Rules

If an IRD file has been registered, this type of rule can be used to define search criterions on value level. The following sub criterions can be set:

IRD Alias Selects the I2C slave, which has been addressed by the I2C master. Only I2C slaves are shown here, for which IRD files have been registered (see section 5.2.2).

Value Value of the I2C slave.

Minimum Minimum for the value, which has been read from the value or which has been written to the value.

Maximum Maximum for the value, which has been read from the value or which has been written to the value.

PEC Some SMBus slaves support ”packet error checking”. For such slaves this field can be used to search for I2C messages with a specific PEC state.

Direction Specifies the direction of the transfer. If ”TX” is specified only transfers to the value are found. If ”RX” is specified only transfers from the value are found.

The rule shown in figure 5.36 will match, when the ”Voltage” value of the ”Smart Battery Data” device at address ”0x0b” is read and the value is between ”10” and ”20”.

Figure 5.37: Status Bar: Saving Trace Data

5.9 Storage of Trace Data

As we have seen in the previous sections there are several formats, to which the traced data can be exported. I2C Studio only supports exporting the data to these formats. It is not possible to read these files back into I2C Studio.

There are several situations where a developer wants to store the traced data and read it back later into I2C Studio. This can be useful e.g. to pass the data to other developers, when working on a problem caused by the I2C communication.

I2C Studio can save the monitored data (I2C messages, analog shots, and transitions of the input testpins) into a proprietary binary file format, which can also be read back. These files are using the extension `*.i2cl`. To store the data efficiently these files are compressed.

To create such a file the user simply clicks on the menu item File|Trace Data|Save As. A file selection dialog gets opened, where the user has to specify the name of the file to be created. After a file name has been specified, the saving starts.

This is done in the background. The current status of the saving is shown in the status bar of I2C Studio, see figure 5.37.

The Cancel button in the status bar allows the user to cancel the saving process, while it is running.

There are three ways to load an `*.i2cl` file into I2C Studio. The developer can use the menu item File|Trace Data|Load or he can simply drag and drop the file from an Explorer window to I2C Studio. Furthermore the user can simply double-click onto the file in the Explorer.

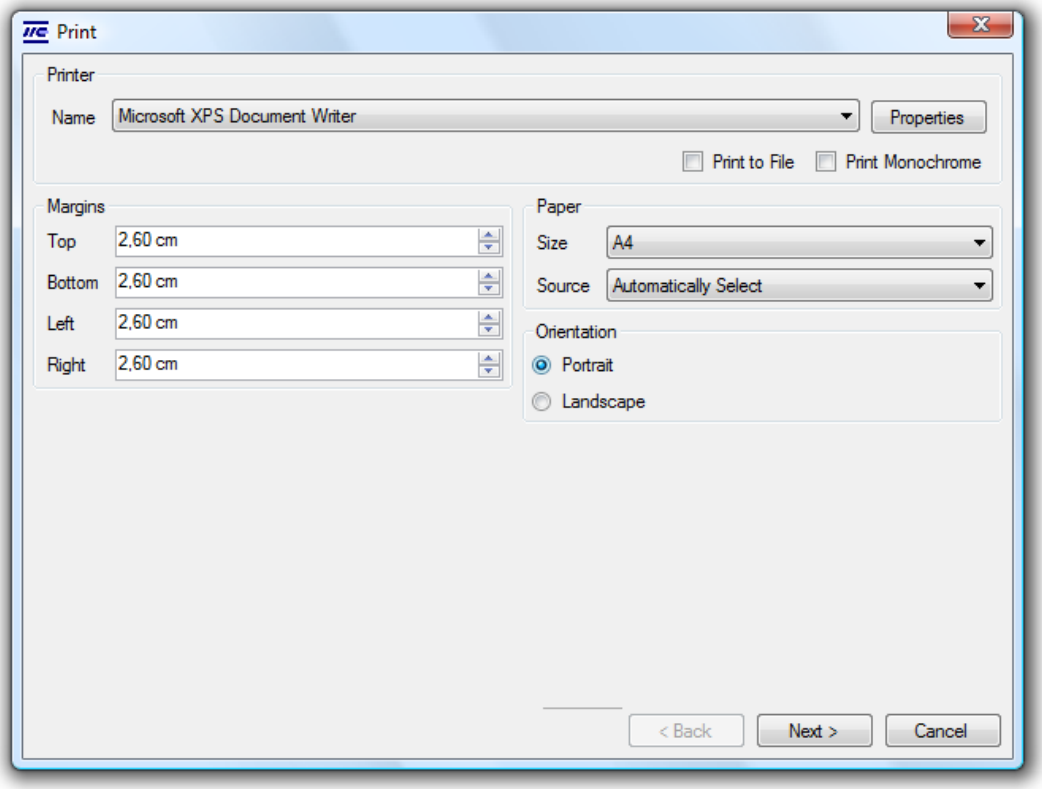


Figure 5.38: Print Wizard: Printer Configuration

5.10 Printing

The data, which has been monitored with I2C Studio, cannot only be written to files but can also be printed.

After some data has been recorded the user can select the menu item File|Print to open the print wizard. This wizard consists of two pages: the first page allows to select and configure the printer, the second page specifies the data to be printed.

5.10.1 Printer Configuration

The first page of the wizard can be seen in figure 5.38. The following parameters can be configured by the user:

Name Selects the printer to be used.

Properties This button opens a printer specific dialog, which can be used to configure printer specific options like e.g. the duplexer of the printer.

Print to File Selects whether the data generated by the printer driver is sent directly to the printer or written into a file.

Margins Defines the top, right, bottom, and left margins.

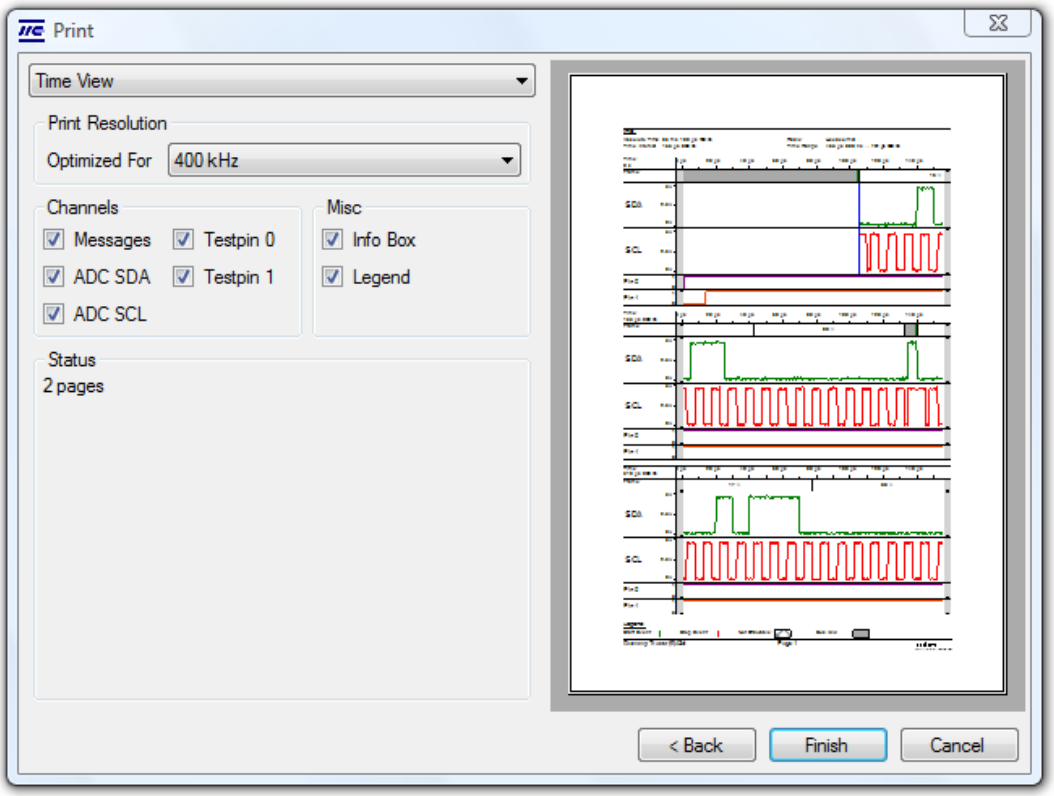


Figure 5.39: Print Wizard: Data Selection

Paper Size Most printers support not only one size of paper but a large quantity of different formats. Selects the format to be used.

Paper Source Larger printer have got more than one paper tray. This option specifies the tray to be used.

Orientation The data can be rendered in the portrait or landscape mode.

After the printer has been configured, the user can jump to the next page of the wizard using the Next > button.

5.10.2 Data Selection

The second page of the print wizard, which can be seen in figure 5.39, can be divided into three major parts.

On the upper left side there is a combo box, which configures the view to be printed. Here the user can select between the time view and the message view.

Under this combo box there are the options, which define the data to be printed.

Finally on the right side there is a preview of the first page to be printed. The preview depends on the currently configured options.

To start the printing the user has to select the Finish button.

Time View

If the user selects to print the time view, all data, which is currently visible in the time view window, gets printed. To print e.g. all traced I2C data, the user has to zoom out completely in the time view window before opening the print wizard.

The following options can be configured when printing the time view:

Optimized For If the user selects to print several I2C messages at the same time, he will always result into print outs, whose resolution is too low to see any interesting details. To solve this problem, I2C Studio offers a functionality to split the data to several pages.

This option configures the number of pages, which are created. To make the user the decision, how many pages are needed, very easy, he can simply specify the needed resolution as frequency.

Let us assume the traced I2C bus uses a clock rate of 400 kHz. For this set-up the user would simply select an "Optimized For" value of 400 kHz. If there are analog shots of the electrical levels available, typically higher frequencies like e.g. 6 MHz should be selected.

Channels Sometimes the user do not want to print out all available parts of the time view. Using these switches the miscellaneous parts can be selected or deselected to be printed.

Info Box A box containing some information can be added to the beginning of each page. This box includes the absolute time, where the data of this page starts, the time interval of the complete page, the radix, and the time range covered by the page.

Legend A legend describing the possible symbols can be added to the end of each page.

Status Contains the number of pages, which will be created, when the user selects to print with the current configuration.

Message View

If the user selects to print out the message view, the following options are available:

Message Range Specifies the I2C messages to be printed out.

Mode Like in the GUI it is possible to choose between three different modes: the raw view, the register view, and the value view.

Columns If the user does not want to print out all possible columns of the table, these switches can be used to disable the appropriate columns.

5.11 Plug-Ins

To get a high-level view of the traced I2C data the user can register IRD files or plug-ins, which convert raw messages to interpreted messages. IRD files, which are based on XML, can be written easily using a normal text editor or the IRD composer of I2C Studio.

They are very good solution for most register based I2C slaves. For more complex I2C high-level protocols there is a better suited solution: the plug-in interface.

Plug-Ins can be written using all programming languages supporting the Microsoft .NET framework. The following example shows how to program a plug-in in C#:

```
using telos.I2cApi.DotNet;
using System;
using System.Collections.Generic;
using System.Text;


public sealed class PlugIn : PlugInInterface
{
  /*********************************************************/
  /// <summary>
  /// This method describes the IC, whose traffic can be
  /// parsed by the plug-in.
  /// </summary>
  /*********************************************************/

  public override PlugInInterface.Device SupportedDevice
  {
    get
    {
      I2cAddressRange        address_range;
      List<I2cAddressRange>  address_ranges;


      // create I2C address range containing I2C addresses, which are
      // supported by this IC
      address_ranges = new List<I2cAddressRange>();
      address_range = new I2cAddressRange (new I2cAddress (0x40, false),
                                           new I2cAddress (0x4f, false));
      address_ranges.Add (address_range);

      // create an object describing the IC
      return new PlugInInterface.Device
                  ("telos",
                   "Example Plug-In",
                   "This plug-in demonstrates how to use the plug-in interface.",
                   address_ranges);
    }
  }


  /*********************************************************/
  /// <summary>
  /// Interprets the I2C message with the specified index.
  /// </summary>
  /*********************************************************/
```

```
  public override string InterpretData (PlugInRawTraceData data, int index)
  {
    ParsedTraceData[]  msg;
    ParsedTraceData    sub_msg;
    StringBuilder      result;


    // create object for the result of the parser
    result = new StringBuilder();

    // get the I2C message to be parsed by the plug-in
    msg = data[index];

    // iterate over the submessages of the I2C message
    for (int i=0; i < msg.Length; i++)
    {
      // new line
      if (i > 0)
        result.Append ("\n");

      // get the submessage
      sub_msg = msg[i];

      // output the direction of the submessage
      switch (sub_msg.Direction)
      {
        case ParsedTraceData.DataDirection.DIRECTION_TX:
          result.Append ("TX: ");
          break;

        case ParsedTraceData.DataDirection.DIRECTION_RX:
          result.Append("RX: ");
          break;

        default:
          continue;
      }

      // output the data bytes of the submessage
      for (int a=0; a < sub_msg.Data.Length; a++)
      {
        if (a > 0)
          result.Append (" ");

        result.AppendFormat ("{0:X2}", sub_msg.Data[a]);
      }
    }

    // return result
    return (result.ToString());
  }
}
```

A complete documentation of the plug-in interface can be found in the I2C.NET API documentation.

6 Master

Beside the tracer function window there is a second important type of function window in I2C Studio: the master window. This type of window can be used to act actively as an I2C master on an I2C bus.

An I2C master can write data to I2C slaves and read data from them. Only I2C masters can initiate transfers on an I2C bus.

6.1 Introduction

The New button or the File|New menu item can be used to open a new master window. Alternatively a new master window can be opened using the context menu of the workspace.

Figure 6.1 shows I2C Studio with a master window, which has just opened. It is completely empty. No commands have been defined yet.

The window is divided into three main areas. On the upper left side there is the command tree. This tree contains all commands, which are executed one by one when the user clicks on the Start button. The upper right side of the window contains detailed information about the command, which is currently selected in the command tree. The remaining space in the lower part of the window is filled out by the result view.

6.2 Creation of Messages

6.2.1 Command Tree

After opening a new master window the developer has to define the messages, which should be transferred over the I2C bus.

Figure 6.2 shows a command tree, which contains five commands grouped into two folders. The first folder contains four commands, which are used to initialize a TV tuner and the second folder contains one command, which changes the currently active TV channel.

Each line in the command tree starts with a check box. If this box is checked, the command or rather the content of the folder is going to be executed when the user clicks on the Start button. Items, whose boxes are not checked, are not going to be executed. This feature can be useful, if the developer wants to deactivate some commands temporarily without deleting them completely from the tree.

The check box is followed by one of the following icons:

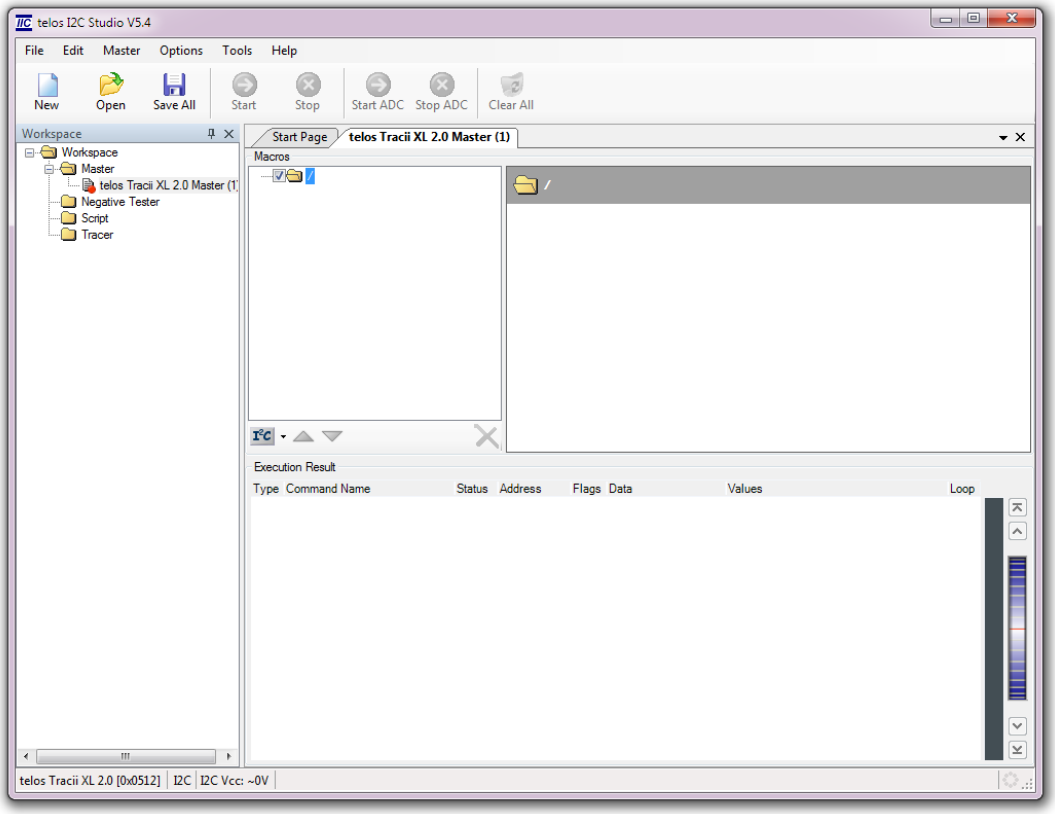


Figure 6.1: New Master Window

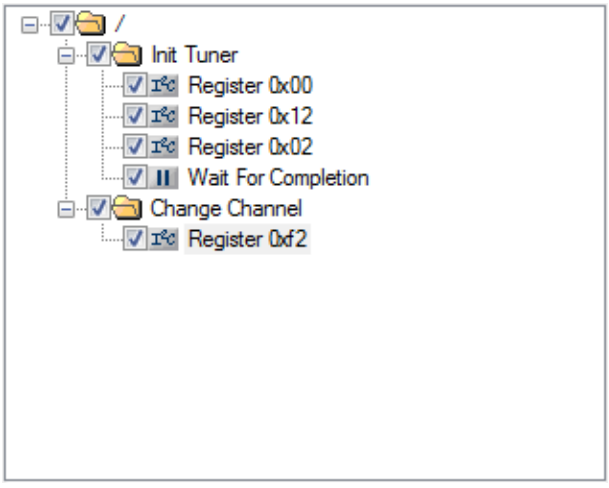


Figure 6.2: Command Tree

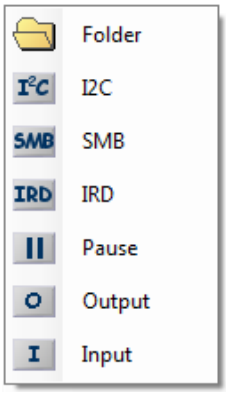


Figure 6.3: Drop-Down List for Changing the Type of the Command

folder (opened)
folder (closed)
I2C command
SMBus command
IRD command
pause command
input testpins command
output testpins command

The last element in each line is a text, which can be entered by the user to identify a command.

There are several different ways, how to fill and modify the command tree. Below the command tree there are four buttons, which can be used for this purpose. They have got the following meaning:

Adds an I2C command to the command tree. There is a drop-down list on the right side of the button, which can be used to change the type of command. Figure 6.3 shows this list.

Moves the selected item upwards in the tree.

Moves the selected item downwards in the tree.

Deletes the selected item from the tree

Instead of the buttons the developer can use the context menu of the command tree. This menu is shown in figure 6.4.

6.2.2 I2C

An I2C command sends an I2C message to an I2C bus. Such an I2C message consists of one or more submessages. The I2C command shown in figure 6.5 consists of two submessages: a

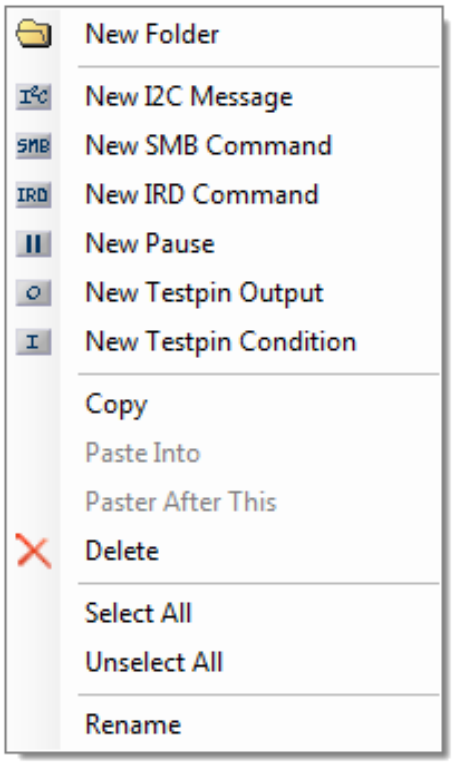


Figure 6.4: Context Menu

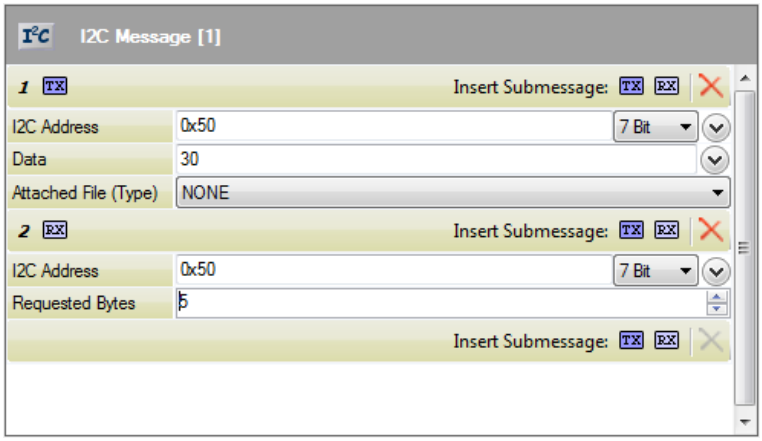


Figure 6.5: I2C Command

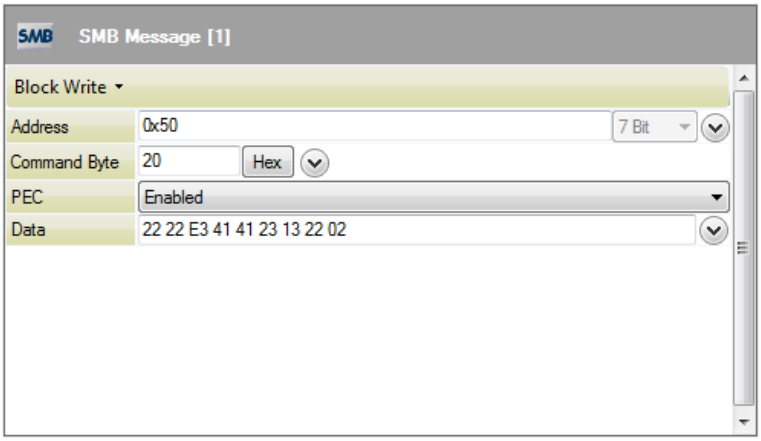


Figure 6.6: SMBus Command

master transmitter submessage followed by a master receiver submessage.

The submessages get numbered serially. The number can be found on the left side of each submessage. This number is followed by one of the following icons:

TX	master transmitter submessage
RX	master receiver submessage

On the right side there are three buttons before and after each submessage:

TX	insert a master transmitter submessage
RX	insert a master receiver submessage
✕	delete this submessage

The submessages contains the following entries:

I2C address I2C address of the slave.

Data Data bytes to be sent to the slave (master transmitter only).

Attached File (Type) This can be set to ”NONE”, ”ASCII HEX”, ”INTEL HEX”, and ”BINARY”. If it is set to a value unequal ”NONE”, data gets read from an external file. This data gets sent after the data, which has been specified in the ”data” field. This option specifies the format of the external file (master transmitter only).

Attached File (Name) Path of the external file (master transmitter only).

Requested Bytes Number of bytes to be read from the slave (master receiver only).

By clicking into the grey headline, it is possible to change the alias name of the command. This feature is not only supported by I2C commands, but by all other commands.

6.2.3 SMBus

While the I2C specification allows I2C messages with an arbitrary number of submessages and arbitrary length of the submessages, the SMBus specification defines a limited set of message types:

- Quick Command
- Send/Receive Byte
- Write/Read Byte
- Write/Read Word
- Process Call
- Write/Read Block
- Block Write - Block Read Process Call

Note: The SMBus command is not supported by all I2C interfaces.

The first step after the creation of an empty SMBus command is the selection of the message type. Afterwards the command has to be configured. Depending on the chosen message type some of the following inputs are displayed:

I2C address I2C address of the slave.

Direction Specifies the direction of the transfer. ”TX” sends data to the slave, ”RX” reads data from the slave.

Command Byte Specifies a command. This informs the slave how to process the received data or which data has been requested by the master respectively.

Data (Byte/Word) Data to be sent to the slave.

PEC Enables or disables the ”packet error checking” (PEC) of the SMBus. PEC adds a CRC8 to each transfer to detect transmission errors.

Figure 6.6 shows the SMBus command mask. In this example the message type has been set to ”Block Write”. The message gets sent to the slave with the address ”0x50”. Packet error checking has been enabled. The data bytes, which gets transmitted to the slave, are ”22 22 E3 41 41 23 13 22 02”.

Note: There are some ICs on the market, whose PEC implementation does not conform to the SMBus specification.

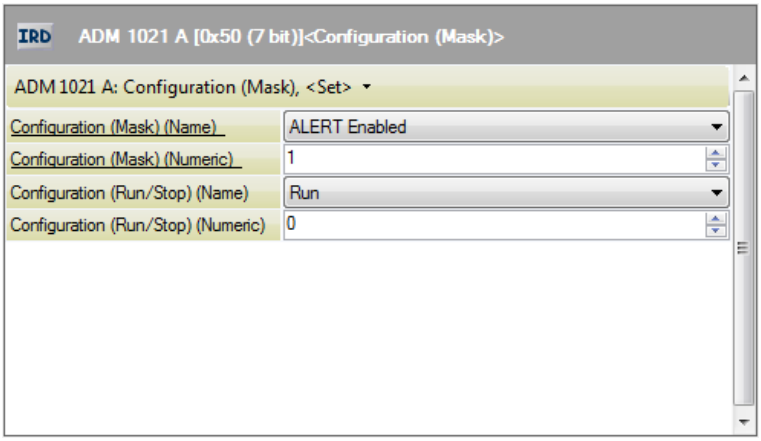


Figure 6.7: IRD Command

6.2.4 IRD

While the I2C and SMBus commands allow to create messages on a very low level, the IRD commands offer a higher abstraction. The I2C Register Description (IRD) defines a file format, which allows to describe the register and values of ICs.

Note: The IRD feature is not supported by all I2C interfaces.

Before an IRD command can be created, it is necessary to use the ”IRD & Plug-In Manager”, see section 5.2.2. In this manager the developer can specify the IRD files, which should be used for the slaves connected to the I2C bus.

After the slave addresses have been registered, the slave and the value to be read or written can be selected. This is done using the drop-down list at the upper part of the IRD command editor. Figure 6.7 shows this editor. In this example the user has selected ”ADM 1021 A: Configuration (Mask), <Set>”. ”ADM 1021 A” is the alias name of the slave, which has been specified in the ”IRD & Plug-In Manager”. ”Configuration (Mask)” is the name of the value to be modified. The direction of the command is specified by ”<Set>”.

If the user has selected to set a value in the IC, the value to be set must be defined in the next step. In this example the value ”Configuration (Mask)” will be set to ”ALERT Enabled”. The content can be specified numerically or by name.

Very often it is not possible to set values separately. Instead a whole group of values must be modified at once. This is caused by the fact that the values are stored in registers. Very often a register stores the content of several values. In this example ”Configuration (Run/Stop)” must be set besides ”Configuration (Mask)”.

The value, which has initially been selected in the drop-down list, gets underlined.

6.2.5 Pause

There are some situations, where the user does not want to send all specified messages as soon as possible to the I2C bus. An example for such a situation are EEPROM or flash ICs. After transferring some data to such an IC, the device needs some time to copy the data from its internal cache to the EEPROM or flash cells.

Figure 6.8: Pause Command

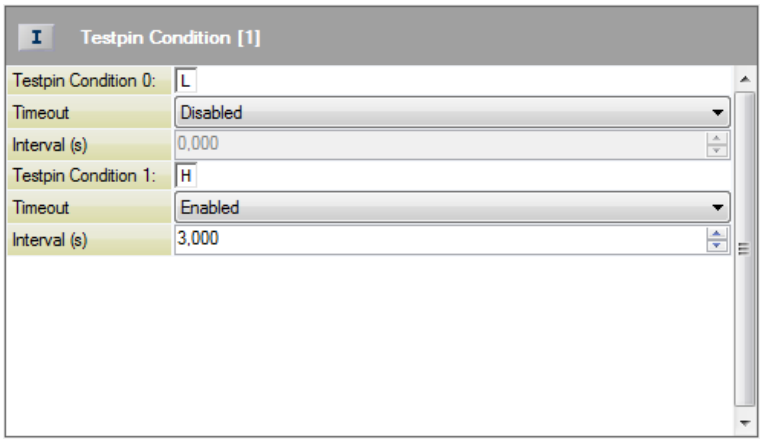


Figure 6.9: Input Testpins Command

A pause command tells I2C Studio to wait the specified time before going on with the next command in the tree.

Figure 6.8 shows the editor for pause commands. In this example the interval of the pause has been set to 5.2 seconds.

Note: The precision of the interval is limited. It is guaranteed that the interval is not shorter than the specified time, but it can be longer. This is not a problem for the typical usage of this feature.

6.2.6 Input Testpins

All I2C interfaces have got a number of digital input lines. This command type can be used to wait for certain levels on one or more lines.

The editor for the input testpins command is shown in figure 6.9. For each available line there are three parameters:

Testpin Condition Defines the level to be expected:

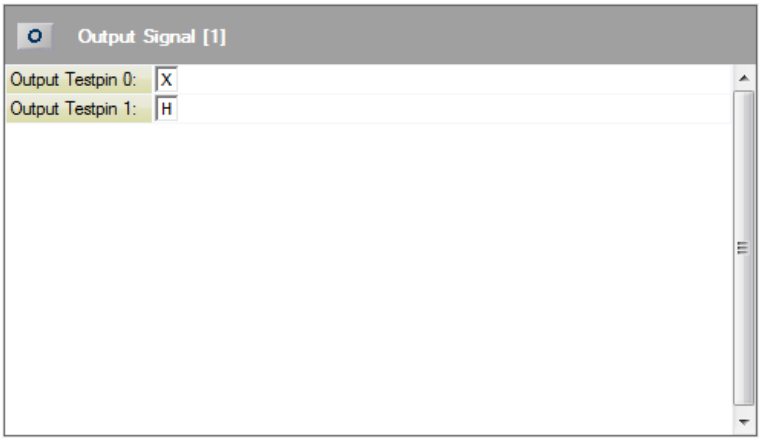


Figure 6.10: Output Testpins Command

L	low level
H	high level
X	don't care

Timeout Specifies whether to use a timeout or not.

Interval (s) If the timeout has been enabled, this parameter defines how long I2C Studio waits for the expected level. If the expected level is not seen within this interval, I2C Studio simply continues.

6.2.7 Output Testpins

With the output testpins command the user can set the level of the digital output lines of an I2C interface. All I2C interfaces have got at least one digital output line.

For each available output testpin there is a line in the output testpins command editor. The I2C interface used for the example in figure 6.10 has got two digital output lines. The user can select one of three possible values for each line:

L	low level
H	high level
X	don't care

In the shown example the level of testpin 0 is set to don't care. So I2C Studio will not change the level of this testpin. Testpin 1 will be set to high level.

6.3 Results

After the user has started the master, the results are displayed in the result view. There is one entry for each command, which has been defined by the user.

The result view consists of a number of columns, which contain the following content:

Type Type of the command.

Command Name The user can assign a name to each command. This name typically describes the purpose of the command.

Status Signals the status of the command. The icons have got the following meaning:

no error

I2C address has not been acknowledged

one of the data bytes has not been acknowledged

error

PEC (packet error checking) error [SMBus only]

Addr Address of the I2C slave, which has been addressed. The icons have got the following meaning:

7-bit I2C address

10-bit I2C address

If an alias name has been registered for the displayed I2C address, this name gets shown in this column.

Flags An I2C message can be transferred into two directions. This column displays which one has been used:

master transmitter

master receiver

master transmitter / master receiver

Data/Values These columns describe the transferred data. The content of both columns is different from command type to command type.

Loop The loop counter shows how often the command has been transferred. The counting starts when the user starts the master e.g. by clicking on the Start button. It ends when the user stops the master e.g. by clicking on the Stop button.

6.3.1 I2C

The Data/Values columns contain the following entries:

Data Data bytes, which have been exchanged with the I2C slave. Data bytes, which have been specified by the user for the master transmitter case and which have not been accepted by the I2C slave, are marked with red color.

Length The first value indicates the number of bytes, which have been exchanged actually with the I2C slave. The second value indicates the number of bytes, which have been specified by the user. If the transfer was successful, both values should be the same.

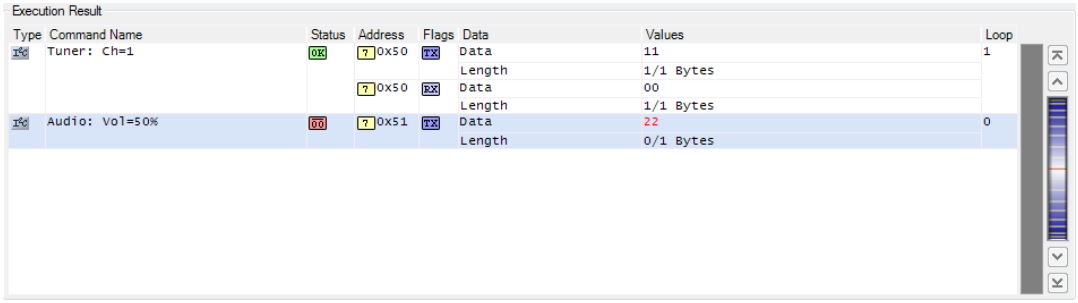

Execution Result
Type Command Name Status Address Flags Data Values Loop
I²C Tuner: Ch=1 OK 0x50 TX Data 11 1
Length 1/1 Bytes
0x50 RX Data 00
Length 1/1 Bytes
I²C Audio: Vol=50% 00 0x51 TX Data 22 0
Length 0/1 Bytes

Figure 6.11: I2C Result

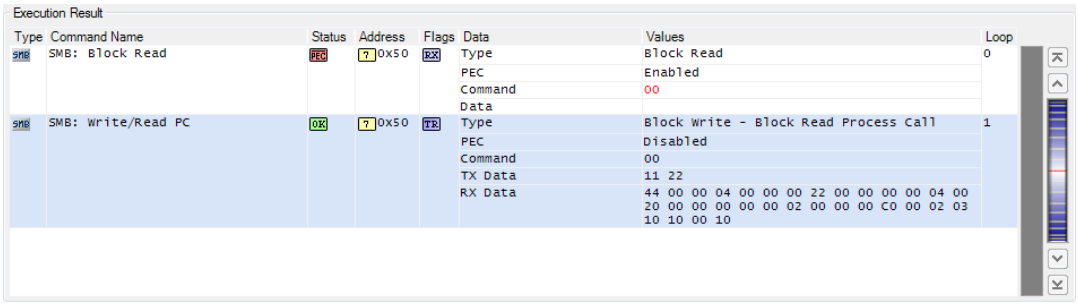

Execution Result
Type Command Name Status Address Flags Data Values Loop
SMB SMB: Block Read PEC 0x50 RX Type Block Read 0
PEC Enabled
Command 00
Data
SMB SMB: Write/Read PC OK 0x50 TX Type Block Write - Block Read Process Call 1
PEC Disabled
Command 00
TX Data 11 22
RX Data 44 00 00 04 00 00 00 22 00 00 00 00 04 00
20 00 00 00 00 00 02 00 00 00 C0 00 02 03
10 10 00 10

Figure 6.12: SMBus Result

The example in figure 6.11 shows two I2C messages. The first message consists of two submessages: a master transmitter submessage containing the data byte ”0x11” and a master receiver submessage requesting one data byte. The message has been exchanged with the I2C slave on address ”0x50”. No error has occurred during the transfer.

The second message is a master transmitter message containing the data byte ”0x22”. The transfer of this message was not successful, because no I2C slave was responsible for the address ”0x51”.

6.3.2 SMBus

The Data/Values columns contain the following entries:

Type Type of the SMBus transfer (e.g. ”Block Read”).

PEC Defines whether the SMBus message has been transferred with or without ”packet error checking”.

Command Command byte, which has been sent to the SMBus slave.

(TX/RX) Data Data bytes, which have been exchanged with the SMBus slave. Data bytes, which have been specified by the user for the master transmitter case and which have not been accepted by the SMBus slave, are marked with red color.

Figure 6.12 shows the result of two SMBus messages. A message of the ”Block Read” type has been transferred as first. The transfer of this message failed, because PEC had been enabled and the PEC checksum has been wrong.

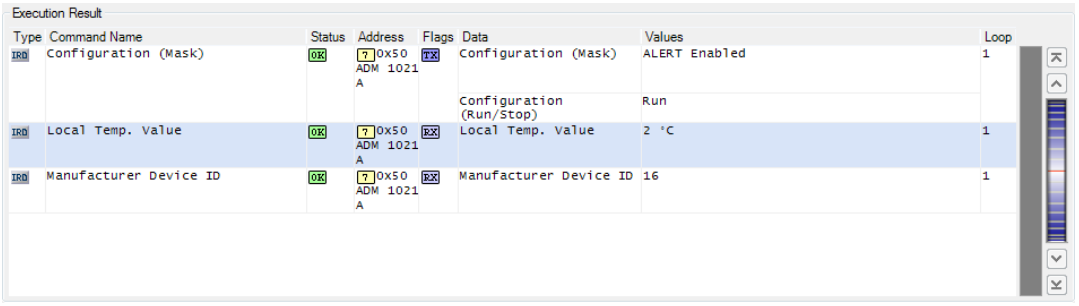

Execution Result
Type Command Name Status Address Flags Data Values Loop
IRD Configuration (Mask) OK 0x50 ADM 1021 A TX Configuration (Mask) ALERT Enabled 1
Configuration (Run/Stop) Run
IRD Local Temp. Value OK 0x50 ADM 1021 A RX Local Temp. Value 2 °C 1
IRD Manufacturer Device ID OK 0x50 ADM 1021 A RX Manufacturer Device ID 16 1

Figure 6.13: IRD Result

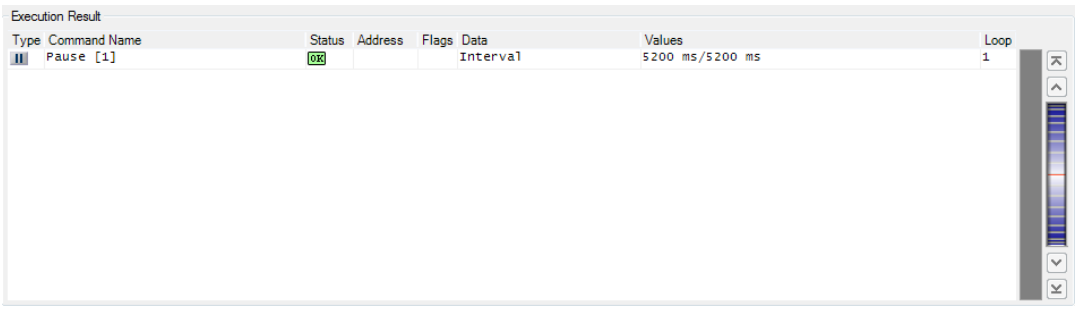


Figure 6.14: Pause Result

The second SMBus message has been transferred successfully. It was of the ”Block Write - Block Read Process Call” type. PEC has been disabled. I2C Studio has sent the command byte ”0x00” and the data bytes ”0x11 0x22”. The SMBus slave answered the request with several data bytes.

6.3.3 IRD

For IRD message the Data column contains the name of the IRD values. The Values column contains the values of the IRD values.

The example in figure 6.13 contains the results for three IRD messages. All messages have been exchanged with an ADM1021A device, which is listening to address ”0x50”.

The first message sets the values ”Configuration (Mask)” to ”ALERT Enabled” and ”Configuration (Run/Stop)” to ”Run”. The second message reads the ”Local Temp. Value” from the device, which is ”2 °C”. The last message reads out the ”Manufacturer Device ID” of the IC, which is ”16”.

6.3.4 Pause

The Data/Values columns contain the following entry:

Interval The first value indicates the time in milliseconds, which has already elapsed. The second value indicates the time, which has been specified by the user.

The example in figure 6.14 shows a pause command. The user has specified an interval of ”5200 ms”. Since the pause has already been completed, both values are ”5200 ms”.

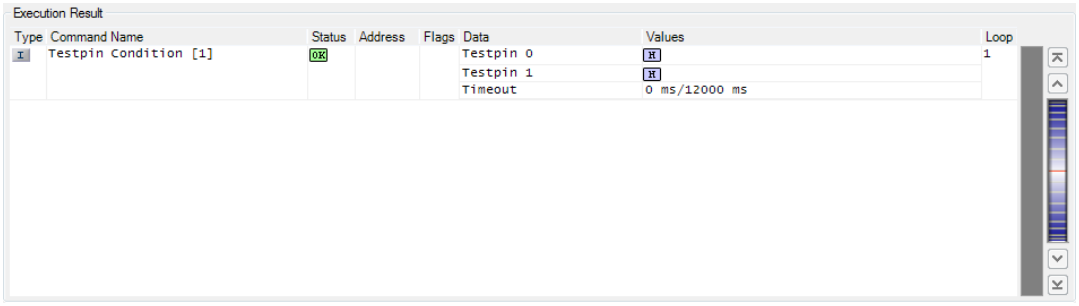

Execution Result
Type Command Name Status Address Flags Data Values Loop
Testpin Condition [1] OK Testpin 0 H 1
Testpin 1 H
Timeout 0 ms/12000 ms

Figure 6.15: Input Testpins Result

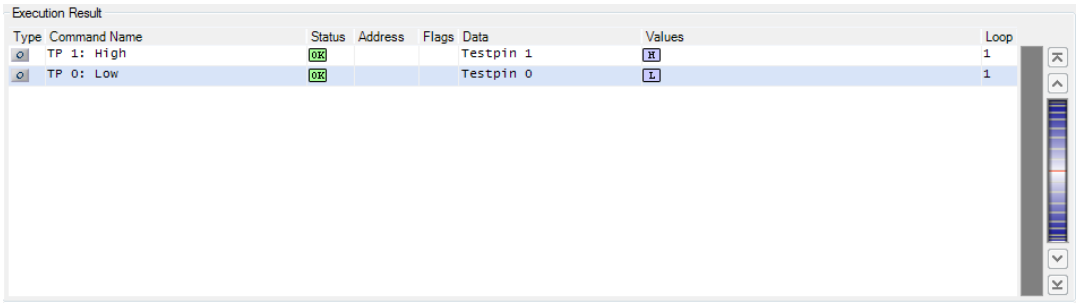


Figure 6.16: Output Testpins Result

6.3.5 Input Testpins

The Data/Values columns contain the following entries:

Testpin (x) Level of the input testpin. The following icons are possible:

H	high level expected & found
L	low level expected & found
H	high level expected, low level found
L	low level expected, high level found

Timeout The first value indicates the time of the timeout, which has already elapsed. The second values indicates the timeout, which has been specified by the user. Both values are given in milliseconds.

The example shown in figure 6.15 contains the result of one input testpins command. The user has specified an input testpins command, which has expected high-levels on testpin 0 and testpin 1. A timeout of ”12 s” has been specified. Both testpin had the expected level, when the master was started.

6.3.6 Output Testpins

The Data/Values columns contain the following entries:

Testpin (x) Level to which the output testpin has been set by the master. The following icons are possible:

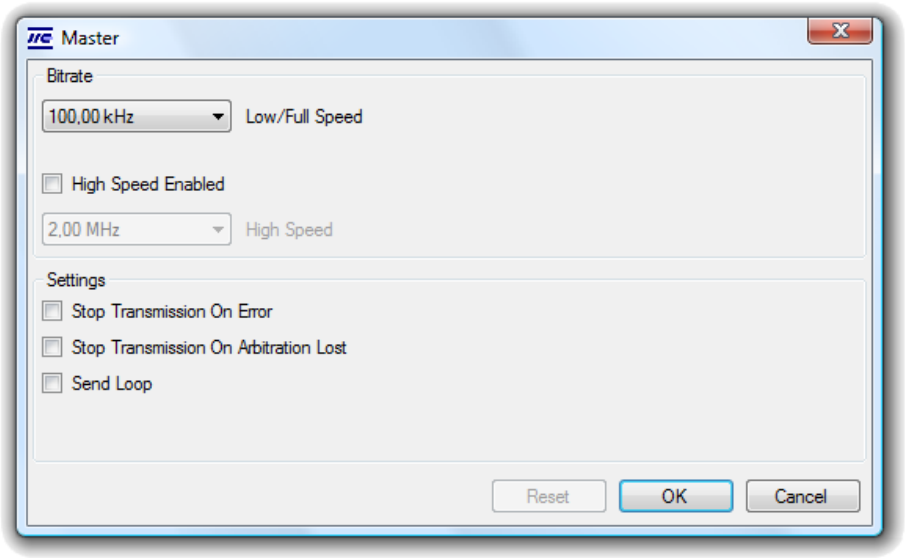


Figure 6.17: Master Options

H	high level
L	low level

The result view in figure 6.16 contains the results of two output testpins commands. Both have been transferred successfully. The first command sets testpin 1 to high level. The second command sets testpin 0 to low level.

6.4 Configuration

6.4.1 Master Options

The master function window can be configurated with its options dialog. The menu item Master|Master Options opens this dialog.

The following parameters can be configured using the master options dialog:

Bitrate Defines the I2C bitrate (SCL) to be used by the master.

Low/Full Speed Configures the low/full-speed bitrate. Some I2C interfaces allow to set I2C bitrates, which do not conform to the values defined by the I2C specification.

High Speed If this value is set, the master sends the messages using the high-speed mode of I2C. This mode is supported only by a few I2C devices on the market. SMBus devices do not support this mode. The high-speed token gets sent with the "Low/Full Speed" bitrate.

Stop Transmission On Error If this check box is set and an error occurs, the master function window goes to the stop state.

Stop Transmission On Arbitration Lost If this check box is set and an arbitration error occurs, the master function window goes to the stop state.

Send Loop Normally the master function window sends each message on time. After sending all messages having been defined by the user, the master goes into the stop state. If this check box is set, the master continues sending the messages until the user stops the master manually.

Note: The I2C high-speed mode is only available for I2C interfaces with a high-speed license.

6.4.2 IRD & Plug-In Manager

The "IRD & Plug-In Manager" dialog gets described in section 5.2.2.

7 Negative Tester

The most common use of an I2C interface's master functionality is to send and receive data in compliance with the I2C specification. This implies that the device we are talking to is known to conform to the I2C standard, as well. However, there are situations in which this particular aspect of a device is to be tested, i.e. where the goal is to determine whether or not a device is fully compliant with the I2C specification and how this device behaves in case of ill-formed signals.

The negative tester is designed to fulfil all requirements introduced by this kind of application. It is capable of acting as master or as slave. In both roles it allows to control all relevant low-level aspects of bus communication, in particular it provides functionality to define the timing for each and every bit. It is even possible to send out patterns which violate the standard, e.g. make a "byte" six bits long. In slave mode the negative tester permits to stretch the clock at any time. The user has full control over arbitration handling in master mode.

Furthermore, independent configurable output signals can be used to trigger external events or to flag certain states in real time.

7.1 Introduction

A negative tester function window can be opened by clicking on the New button and selecting a negative tester. Another way is to click on the File|New menu item. Besides it is possible to open a new window using the context menu of the workspace.

Note: The negative tester function window is only available, if a Negative Tester interface is connected to the local or a remote PC. Besides the telos Negative Tester a telos Tracii XL or telos Tracii XL 2.0 with a tracer license is needed.

The Negative Tester and the telos Tracii XL / telos Tracii XL 2.0 must be connected to the same I2C bus. The Negative Tester acts as a pattern generator and the telos Tracii XL / telos Tracii XL 2.0 as I2C tracer.

Figure 7.1 shows a negative tester function window, which has just been opened. The window is divided into three regions. The highest region contains the configuration of the test cases for the master and the slave mode.

This region is followed by the global configuration. Parameters, which can be configured in this region, apply not only to one special test case but to all test cases.

The last region contains the results of an negative tester run. After opening the negative tester function window this region is empty. When the negative tester has been started the first time, the user can browse through the results using this region.

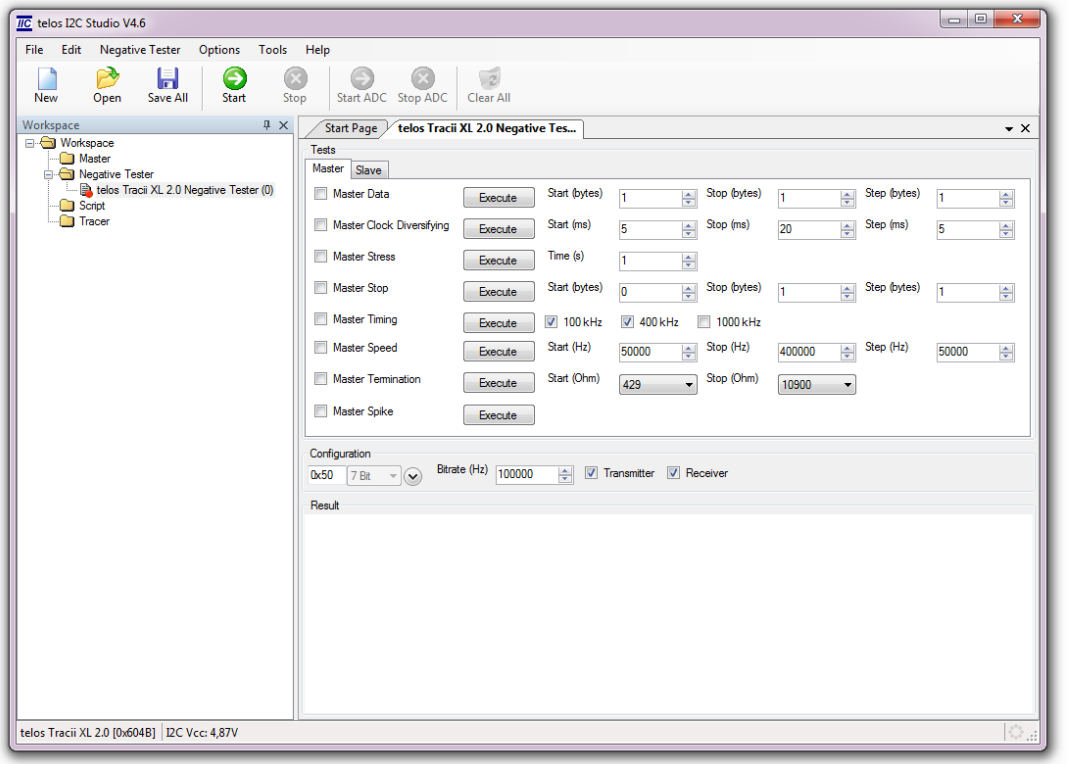


Figure 7.1: New Negative Tester Window

Figure 7.2: Negative Tester and telos Tracii XL 2.0

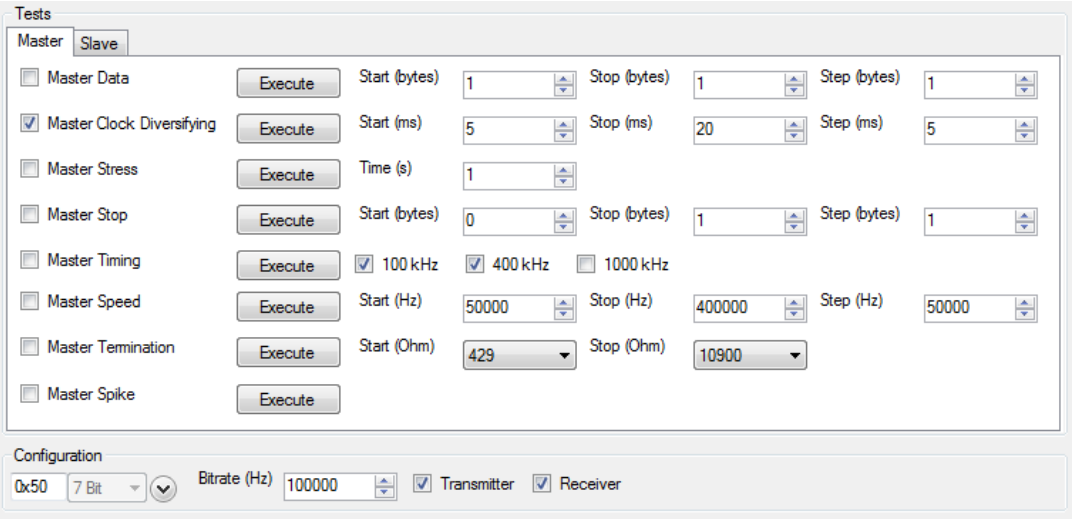


Figure 7.3: Configuration: Master

7.2 Test Cases

The negative tester supports two different roles. It can work as an I2C master or as an I2C slave. If the ”Master” tab page is visible, the negative tester acts as I2C master. To work as an I2C slave, the user has to select the ”Slave” tab page.

7.2.1 I2C Master

As one can see in figure 7.3 there are several different tests available, where the negative tester works as an I2C master. The user can select the tests to be run by using the check boxes in the first column of the configuration. All selected tests are executed, when the developer clicks on the Start button or selects the Negative Tester|Start item from the menu.

If the user wants to simply execute one test, he can use the Execute buttons in the second column of the table.

The third column of the table contains the parameters for the test cases.

Master Data

This test case sends messages with different lengths to the I2C bus. The first message, which gets transferred on the I2C bus, has got the length specified by ”Start (bytes)”. The length of the next messages get incremented by ”Step (bytes)”. If the length exceeds the value specified by ”Stop (bytes)”, the test stops.

Note: A master receiver transfer with a length of zero bytes works only with special I2C slaves. Other I2C slaves can hold the SDA line low after this test has been executed.

Master Clock Diversifying

This test case transfers I2C messages to the I2C bus, which contain longer pauses between some of the bits. Using this test it is possible to check whether a slave works correctly with an asynchronous clock. These pauses are added between different positions of the bits.

The length of the pauses can be configured by the user. The first message uses the pauses specified by ”Start (ms)”. The pauses of the next messages are incremented by ”Step (ms)”. This test stops, after the length has been reached, which is specified in ”Stop (ms)”. All values are specified in milliseconds.

Master Stress

A lot of problems will only occur, when there is a lot of traffic on the I2C bus and the I2C slave on test has to handle a lot of requests within a short time.

This test case is perfect for this situation. For the time specified by the ”Time (s)” value it creates nearly 100% load on the I2C bus. Even the time between two consecutive I2C messages is very short, because the complete message generation is handled in the Negative Tester interface. So the PC does not add any latency.

Master Stop

This test case transfers I2C messages to the I2C bus, which contain STOP conditions at illegal positions. E.g. a message like ”START, ADDR, 5 Data Bits, STOP” gets created. The purpose of this test is to verify that an I2C slave resets its internal state machine correctly after receiving a STOP condition.

Message lengths within the range specified by ”Start (bytes)” and ”Stop (bytes)” are tested. The length gets increment by ”Step (bytes)”.

For each message length to be tested eight transfers are created. Each transfer has got the STOP condition after a different bit position. The first transfers contains the STOP condition after the first bit of the last byte, the second transfers after the second bit of the last byte and so on.

Master Timing

This test case transfers an I2C message to the I2C bus with different timings. Several combinations of $t_{HD;DAT}$, $t_{SU;DAT}$, and t_{HIGH} are tested. The tested minimum and maximum values conform to the value specified for standard- and fast-mode devices in (see section 15.1):

> The I2C-Bus Specification
> Version 2.1
> January 2000

The values for fast-mode plus devices are specified in:

> I2C-bus specification and user manual
> Rev. 03
> 19 June 2007

Using the check boxes it is possible to execute this test only for the requirements of standard-mode, fast-mode or fast-mode plus devices or for all modes.

Note: For this test the signal slope on SCL and SDA is important, because otherwise it cannot be guaranteed that the negative tester can create the requested timings on the I2C bus. Due to this fact it is important to use a strong termination on the I2C bus and to prevent long cables.

Master Speed

This test case transfers an I2C message with different bitrates. The purpose of this test is to verify that an I2C slave supports the specified bitrates.

The first message uses a bitrate like specified by ”Start (Hz)”. The following messages are using bitrates increment by ”Step (Hz)”. This test case ends when the bitrate has reached the value given by ”Stop (Hz)”.

Master Termination

This test case transfers an I2C message with different values for the I2C bus termination (SCL/SDA). The purpose of this test is to verify that an I2C slave works correctly with different types of slopes.

The first message uses a termination like specified by ”Start (Ohm)”. The following messages are using the next values from the list of possible terminations. This test case ends when the value for the termination has reached the value given by ”Stop (Ohm)”.

Master Spike

This test case adds spikes to the transferred I2C message. Conforming to the I2C specification an I2C slave has to suppress spikes of up to t_{SP}.

Note: For this test the signal slope on SCL and SDA is important, because otherwise it cannot be guaranteed that the negative tester can create the requested timings on the I2C bus. Due to this fact it is important to use a strong termination on the I2C bus and to prevent long cables.

General Options

There are some options, which are common for all master test cases. These options can be found in the ”Configuration” region.

The first parameter specifies the I2C address of the slave device, which should be tested by the test cases.

The second parameter is the bitrate. Most messages will be transferred using this bitrate. Only some test cases, which are testing the bitrate itself, may use other bitrates.

The negative tester can work as master transmitter and/or master receiver. Both directions can be enabled/disabled using the ”Transmitter” and ”Receiver” check boxes.

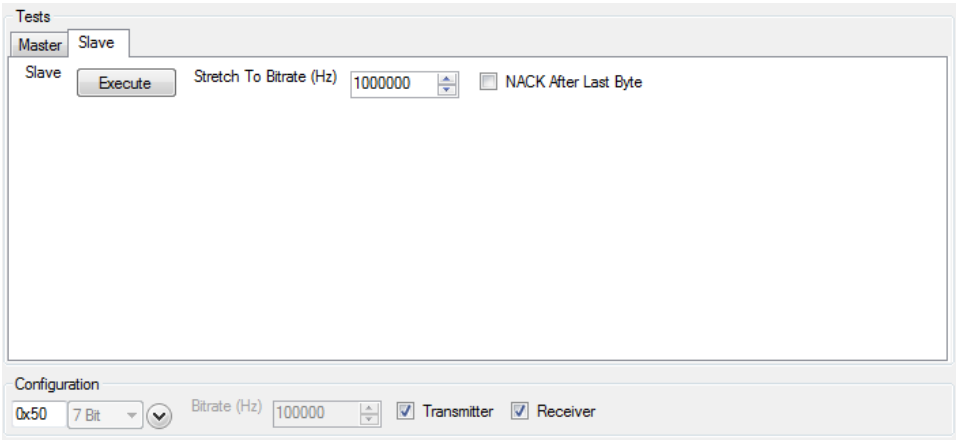


Figure 7.4: Configuration: Slave

7.2.2 I2C Slave

The negative tester can be used not only to test I2C slaves, but also to test I2C masters. To test an I2C master it is necessary, to switch the negative tester to the I2C slave mode. Figure 7.4 shows the activated I2C slave mode.

There are not so many parameters in this mode, which can be configured by the user.

The "Stretch To Bitrate" value specifies the frequency to which the slave stretches the clock. If the master e.g. sends with 400 kHz a value of 20 kHz will slow down the transfer to 20 kHz. So this option can be useful to test, whether the I2C master's stretching detection works correctly.

Normally the slave receiver will accept more data bytes than have been specified. If the "NACK After Last Byte" option has been enabled, the slave receiver sends a NACK after receiving the next byte.

In the "Configuration" region there are two parameters, which can be configured for the I2C slave mode. The first parameter defines the I2C address, which the emulated I2C slave in the negative tester is listening to.

The negative tester can work as slave transmitter and/or slave receiver. The check boxes "Transmitter" and "Receiver" enable/disable these modes.

7.3 Options

Like the other function windows the negative tester function window has got an options dialog. This dialog can be opened using the Negative Tester|Negative Tester Options menu item. Figure 7.5 shows this dialog.

The following options can be configured within this dialog:

Select Board The telos Negative Tester interface itself works only as I2C master or I2C slave on the bus. Additionally a telos Tracii XL or telos Tracii XL 2.0 interface with tracer license is needed. This interface has to be connected to the same bus like the Negative Tester. With this parameter the telos Tracii XL or telos Tracii XL 2.0 to be used can be selected.

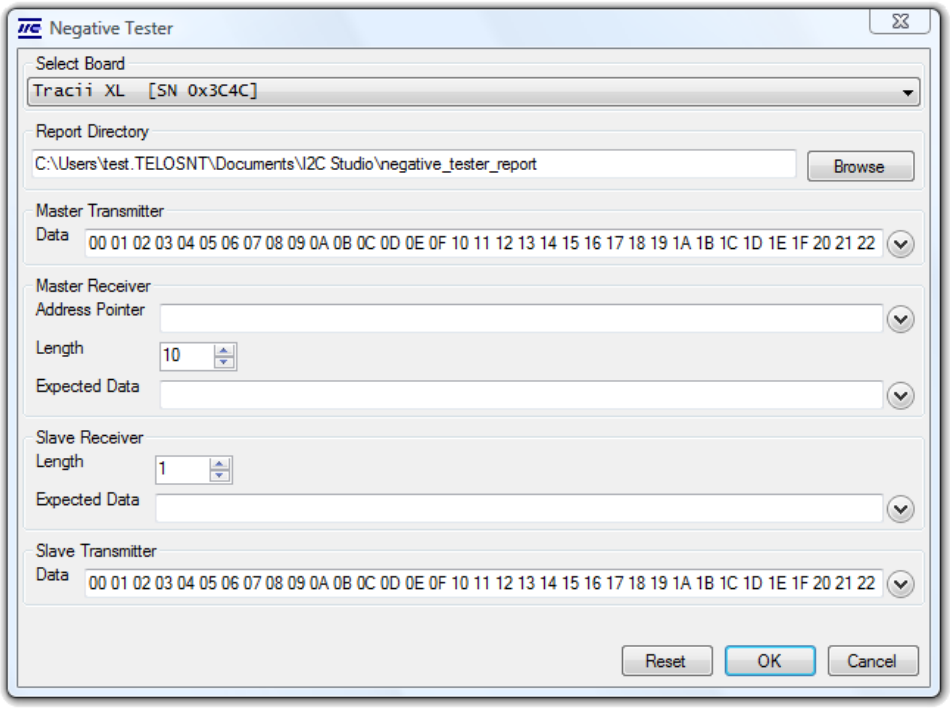

Negative Tester
Select Board
Tracii XL [SN 0x3C4C]
Report Directory
C:\Users\test.TELOSNT\Documents\I2C Studio\negative_tester_report
Browse
Master Transmitter
Data
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
Master Receiver
Address Pointer
Length
10
Expected Data
Slave Receiver
Length
1
Expected Data
Slave Transmitter
Data
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
Reset
OK
Cancel

Figure 7.5: Negative Tester Options

Report Directory During the test run a test report in the HTML format gets generated. This parameter specifies the directory, into which this report gets written.

Master Transmitter The negative tester will use these data bytes for the I2C messages, which gets transferred as master transmitter. Some test cases will not use all bytes. If a test case needs more data than has been specified by the user, the same data will be used several times within an I2C message.

Master Receiver The following parameters can be specified for the master receiver mode:

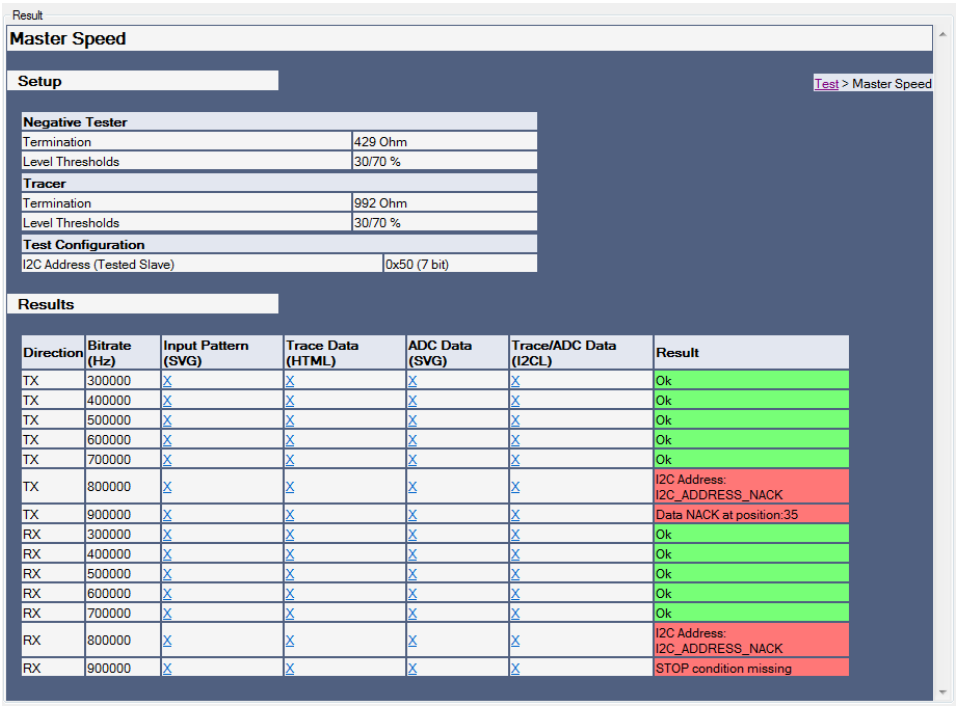
Address Pointer Most I2C devices contain an address pointer, which must be set before reading data from the device. If this field contains any data bytes, the negative tester creates combined transfers instead of simple master receiver transfers on the I2C bus. So each I2C message gets prefixed by an I2C submessage, which sends the data for the address pointer to the device as master transmitter.

Length Specifies the number of bytes to be read from the I2C slave.

Expected Data Defines the data, which is expected to be read from the I2C slave. If the actual read data does not match the data given here, the test case fails.

Slave Receiver For the slave receiver mode the following parameters are available:

Length Configures the number of data bytes, which the slave receiver wants to receive. If the ”NACK After Last Byte” option in the function window is enabled, the negative tester will not acknowledge the last byte.


Result

Master Speed

Setup

Test > Master Speed

Negative Tester	
Termination	429 Ohm
Level Thresholds	30/70 %
Tracer	
Termination	992 Ohm
Level Thresholds	30/70 %
Test Configuration	
I2C Address (Tested Slave)	0x50 (7 bit)

Results

Direction	Bitrate (Hz)	Input Pattern (SVG)	Trace Data (HTML)	ADC Data (SVG)	Trace/ADC Data (I2CL)	Result
TX	300000	X	X	X	X	Ok
TX	400000	X	X	X	X	Ok
TX	500000	X	X	X	X	Ok
TX	600000	X	X	X	X	Ok
TX	700000	X	X	X	X	Ok
TX	800000	X	X	X	X	I2C Address: I2C_ADDRESS_NACK
TX	900000	X	X	X	X	Data NACK at position:35
RX	300000	X	X	X	X	Ok
RX	400000	X	X	X	X	Ok
RX	500000	X	X	X	X	Ok
RX	600000	X	X	X	X	Ok
RX	700000	X	X	X	X	Ok
RX	800000	X	X	X	X	I2C Address: I2C_ADDRESS_NACK
RX	900000	X	X	X	X	STOP condition missing

Figure 7.6: Negative Tester Result

Expected Data Specifies the data, which the negative tester expects to receive from the I2C master. If the data received during the test run does not match this data, the test case fails.

Slave Transmitter Contains the data bytes, which will be sent to the I2C master when working as slave transmitter. If the I2C master requests more bytes than given by this parameter, the negative tester will send the same data bytes several times.

7.4 Result

After the test cases have been configured, the test run can be started using e.g. the Start button of I2C Studio. When the test run is completed, the results get displayed in the result viewer, see figure 7.6.

This viewer is based on Microsoft's Internet Explorer. The report consists of several files, which are connected by links. By clicking on the links it is possible to navigate to the different files. The context menu item Back navigates back to the previously viewed file.

For each test case one report file gets created. Figure 7.6 shows the result for the "Master Speed" test case.

The setup part of the report gives some details about the configuration. In this example e.g. the negative tester and the tracer have both been configured to a termination of "429 Ohm" and level thresholds of "30/70%". The tested I2C slave uses the I2C address "0x50".

The result part of the report contains links to several files:

Input Pattern (SVG) Pattern loaded into the negative tester. This file uses the SVG vector graphics format.

VHDL Stimuli (VHD) Stimuli file for VHDL, which uses the same transitions and timings as generated by the negative tester. This can be used to reproduce errors with a VHDL simulator.

Trace Data (HTML) I2C messages, which have been traced on the I2C bus during the test run. This file uses the HTML format.

ADC Data (SVG) Analogue shot of the I2C lines, which has been captured during the test run. This file uses the SVG vector graphics format.

Trace/ADC Data (I2CL) Contains the same data as ”Trace Data (HTML)” and ”ADC Data (SVG)”. I2C Studio’s own log file format I2CL is used to store this data.

The last column of the table contains the results of the test run. In the given example the I2C device has failed to communicate at speeds higher than 700 kHz.

8 Scripting

8.1 Introduction

The master and the tracer windows provide a powerful and convenient way to use the I2C interfaces from telos. But sometimes users need a more flexible way to access the I2C interfaces. Therefor I2C Studio offers the script window.

A new script window can be created using the New button, the menu item File|New or the context menu item New from the workspace. In contrast to the other window types a script window is never associated to a certain I2C interface. The user has to set up an interface connection manually by calling the appropriate functions from I2C.NET API.

The script window consists of two text boxes. The upper text box allows to enter the script. The lower text box receives error messages from the compiler and the output generated by the script.

The programming language supported by the script window is C#. Two frameworks are available in the script window: .NET 4.0 and the I2C.NET API. The user can add additional libraries, which are based on .NET 4.0.

The programming language C# is defined in ECMA-334, which can be downloaded from here:

```
http://www.ecma-international.org/publications/standards/Ecma-334.
htm
```

The documentation for the I2C.NET API is part of the I2C Studio setup package. It can be opened by the menu item Help|I2C Api Documentation.

I2C Studio comes with APIs for several other languages:

- C++
- .NET (e.g. C++, Visual Basic, and Delphi)
- C
- Labview
- NXP (formerly Philips Semiconductors) URT and URD

These APIs cannot be used directly in I2C Studio. Instead the programmer can use them within the normal development environments.

8.2 Example

The following C# source code can be used to demonstrate the script window:

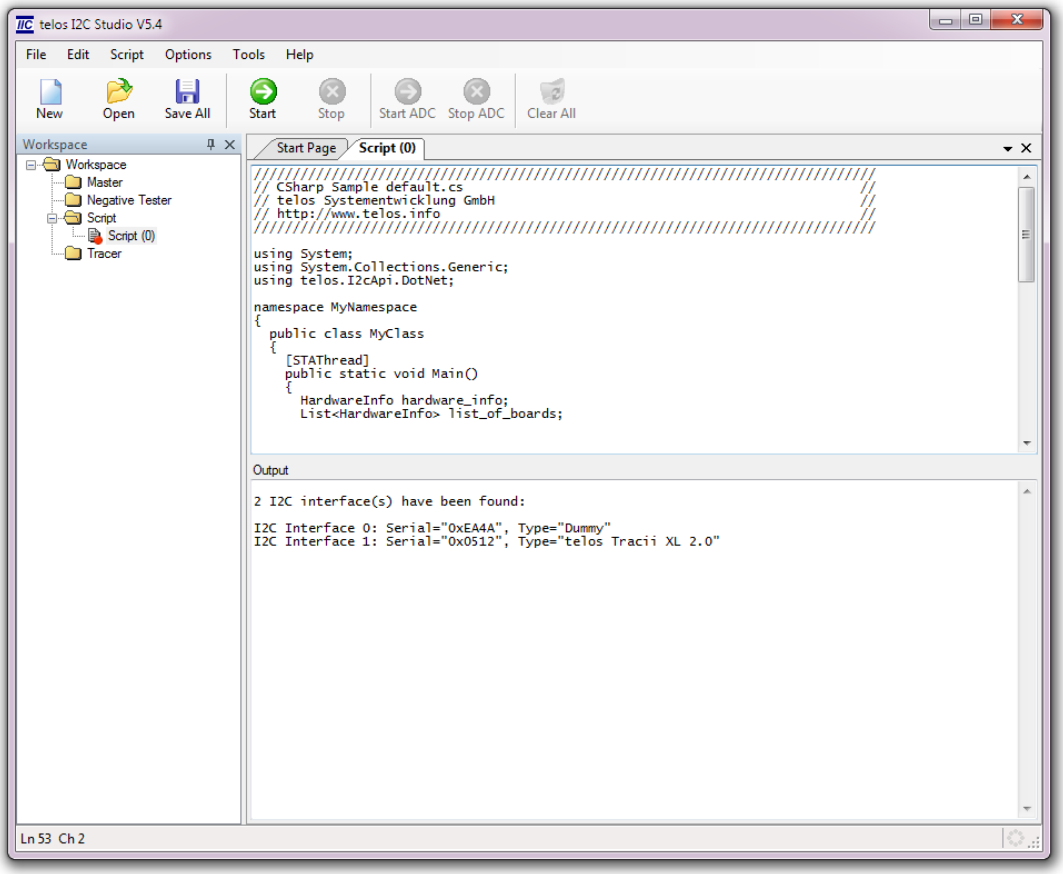


Figure 8.1: Script Window

```
////////////////////////////////////////////////////////////////////////////
// CSharp Sample default.cs                                               //
// telos Systementwicklung GmbH                                           //
// http://www.telos.info                                                  //
////////////////////////////////////////////////////////////////////////////

using System;
using System.Collections.Generic;
using telos.I2cApi.DotNet;

namespace MyNamespace
{
  public class MyClass
  {
    [STAThread]
    public static void Main()
    {
      HardwareInfo hardware_info;
      List<HardwareInfo> list_of_boards;


      try
      {
        using (Board board = new Board())
        {
          list_of_boards = board.ListOfBoards;

          Console.WriteLine();
          Console.WriteLine("{0} I2C interface(s) have been found:", list_of_boards.Count);
          Console.WriteLine();

          for (int i = 0; i < list_of_boards.Count; i++)
          {
            hardware_info = list_of_boards[i];

            Console.WriteLine("I2C Interface {0}: Serial=\"0x{1:X4}\", Type=\"{2}\"",
                              i,
                              hardware_info.SerialNumber,
                              HardwareInfo.GetBoardTypeAsString(hardware_info.BoardType));
          }

          //
          // TODO: Add your script code
          //
        }
      }
      catch (System.Exception ex)
      {
        Console.WriteLine("Error: " + ex.Message);
      }
    }
  }
}
```

This program establishes a connection to the I2C scheduler and outputs a list of the I2C interfaces, which are currently connected to the local PC.

After a script has been entered, it can be started using the Start button or the menu item

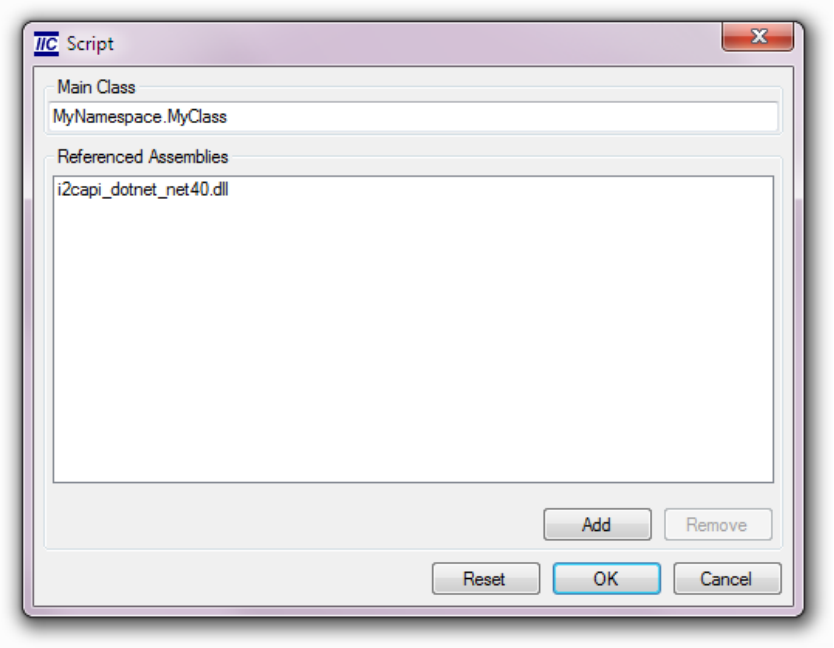


Figure 8.2: Script Window: Options Dialog

Script|Start.

While a script is running, its output is redirected to the output text box.

8.3 Configuration

Like the other function windows the script window has got a configuration dialog, see figure 8.2. This dialog can be opened using the menu item Script|Script Options.

The dialog offers the following options:

Main Class Defines the name of the class, which contains the "Main" method. If the source code uses a namespace, this has to be specified, too. For the example the correct value would be "MyNamespace.MyClass".

Referenced Assemblies Contains a list of .NET assemblies, which should be linked to the executable. To communicate with the I2C interfaces the `i2capi_dotnet_net40.dll` is needed.

Assemblies can be added by using the Add button and removed by using the Remove button.

8.4 Advanced Examples

The setup package of I2C Studio contains some advanced examples, which demonstrate the features of the I2C.NET API. They are located in the installation directory of I2C Studio under `examples\dotnet\`.

The available examples are:

AdcExample.cs Demonstrates the usage of the module, which can create analog shots of the I2C lines SCL and SDA.

InputTestpinsExample.cs Demonstrates the module, which can monitor the transitions of the input testpins.

MasterExample.cs Demonstrates how to send messages on the I2C bus.

NegativeTesterExample.cs Demonstrates the Negative Tester interface.

PlugInExample.cs Demonstrate the writing of plug-ins for I2C Studio.

TracerExample.cs Demonstrate how to record the messages, which have been sent on the I2C bus.

8.5 I2C.NET API (IRD)

Communicating with I2C slaves from a script can be exhausting, because the device specific high-level protocols must be implemented by hand using the data sheets of the devices.

To make life easier I2C Studio comes with the `i2capi_dotnet_ird.dll` assembly. It has been created using the IRD compiler, which is described in section 9.5.

All IRD files of I2C Studio have been converted to .NET classes. These classes allow a type-safe access to the registers and values of several widely used I2C slaves.

The following example demonstrates how to use one of theses classes to read out the Maxim DS 1307, which is connected to the virtual I2C bus of the Dummy I2C interface. The emulated I2C slave uses address 0x68.

```
using System;
using System.Collections.Generic;
using System.Text;
using telos.I2cApi.DotNet;
using telos.Ird.Maxim;


namespace Test
{
  class Program
  {
    static void Main(string[] args)
    {
      Board board;
      Ds1307 clock;


      board = new Board();
      try
      {
        // open Dummy
        board.Open (board.ListOfBoards[0]);

        // create DS 1307 wrapper for address 0x68
        clock = new Ds1307(board, new I2cAddress(0x68, false), "..\\ird");
```

```
        // read out date/time every 500 ms
        while(true)
        {
          Console.WriteLine ((clock.Year+2000).ToString().PadLeft(4) + "/" +
                             clock.Month.ToString("D2") + "/" +
                             clock.Date.ToString("D2") + "  " +
                             clock.Hours.ToString().PadLeft(2) + ":" +
                             clock.Minutes.ToString("D2") + ":" +
                             clock.Seconds.ToString("D2"));
          System.Threading.Thread.Sleep(500);
        }
      }
      finally
      {
        board.Dispose();
      }
    }
  }
}
```

To execute this script in the script window it is necessary to add two .NET assemblies to the options dialog: `i2capi_dotnet_net40.dll` and `i2capi_dotnet_ird.dll`.

The example reads out the data and time from the DS 1307 every 500 ms.

9 Tools

9.1 I2C Memory Slave

The I2C interface types telos Tracii XL and telos Tracii XL 2.0 can be configured to act as a slave on the I2C bus. This slave behaves like a standard I2C RAM with a capacity of 256 bytes. A description of the high-level I2C protocol used by such RAMs can be found e.g. in the data-sheet of the NXP PCF8570 IC:

`http://www.nxp.com/pip/PCF8570_4.html`

The slave in telos Tracii XL 2.0 supports 7-bit and 10-bit I2C addresses. It also supports the I2C high-speed mode.

The configuration dialog (see figure 9.1) for the slave mode is opened by the menu item Tools|I2C Memory Slave. Within this dialog the following options can be configured:

Select I2C Interface This combo box contains a list of I2C interfaces, which can be used as I2C slaves. The configuration shown under "Slave Configuration" refers to the I2C interface selected in this combo box.

Enable If this check box is set, the slave emulation is running.

All I2C Addresses If this check box is set, the emulated RAM listens on all possible 7-bit or 10-bit I2C addresses.

I2C Address Specifies the I2C slave address of the emulated RAM. All I2C requests sent on the I2C bus to this address will be answered by the I2C interface.

Memory Size Configures the size of the emulated I2C slave.

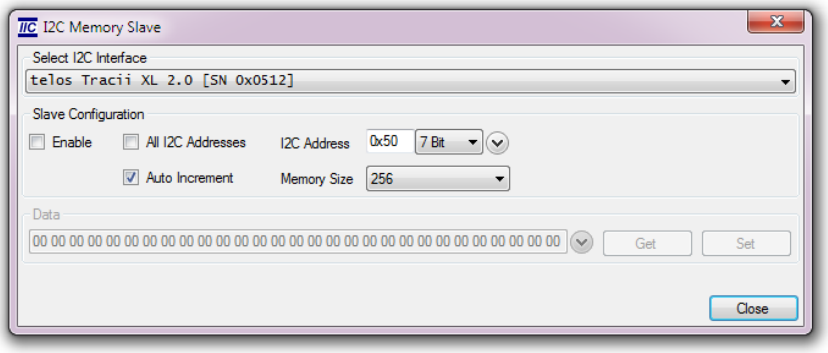


Figure 9.1: I2C Memory Slave

Auto Increment Enables/disables auto incrementing of the address pointer in the emulated I2C slave.

Data Gets/sets the data of the emulated RAM.

Note: telos Tracii XL cannot operate as I2C master and I2C slave in parallel. While the RAM emulation is running, no I2C message can be sent as master from the same I2C interface.

If the user has enabled one or more slaves using the configuration dialog, they stay enabled while I2C Studio is running. Even closing the configuration dialog will not stop the slave emulation.

9.2 VESA DDC

I2C Studio supports the access to VESA DDC EEPROMs that are compliant to the VESA E-EDID 1.3 standard. Such EEPROMs are integrated into all CRT and TFT monitors, which have been sold during the last years. Even modern flat TVs come with VESA DDC support.

These EEPROMs are using the I2C bus. With the aid of these EEPROMs e.g. the PC can configure its graphic card automatically to match the parameters like refresh rate to the display.

I2C Studio comes with a tracer plug-in to interpret I2C messages between an I2C master and a VESA DDC EEPROM.

Furthermore there is a VESA DDC dialog, which can be opened by clicking on the Tools|VESA DDC menu item. This dialog, which is shown in figure 9.2, allows reading, writing, and manipulating the information stored in a VESA DDC EEPROM. So this is quite useful for manufacturers of displays or developers, which have to add display support to their products.

VESA DDC EEPROMs must have a capacity of 128 bytes and must be assigned to the I2C slave address 0x50.

When the user clicks on the Read button, the VESA DDC EEPROM connected to the I2C bus of the selection I2C interface gets read out. Its current content is visualized in the fields of the dialog.

Modified content can be written back to the EEPROM by clicking onto the Write button.

It is not only possible to read and write the content from or to the EEPROM, but it is also possible to import and export the content as a binary image. This is done by clicking onto the Import or Export buttons.

9.3 JEDEC SPD

Besides the VESA DDC EEPROMs there is another important field, where I2C EEPROMs are used in the PC industry: memory modules. All memory modules sold today contain an EEPROM, which describes the timing parameters needed by the module so that the user must not enter these parameters manually. The JEDEC standards and specifications can be downloaded from the JEDEC WWW page at:

```
http://www.jedec.org/
```

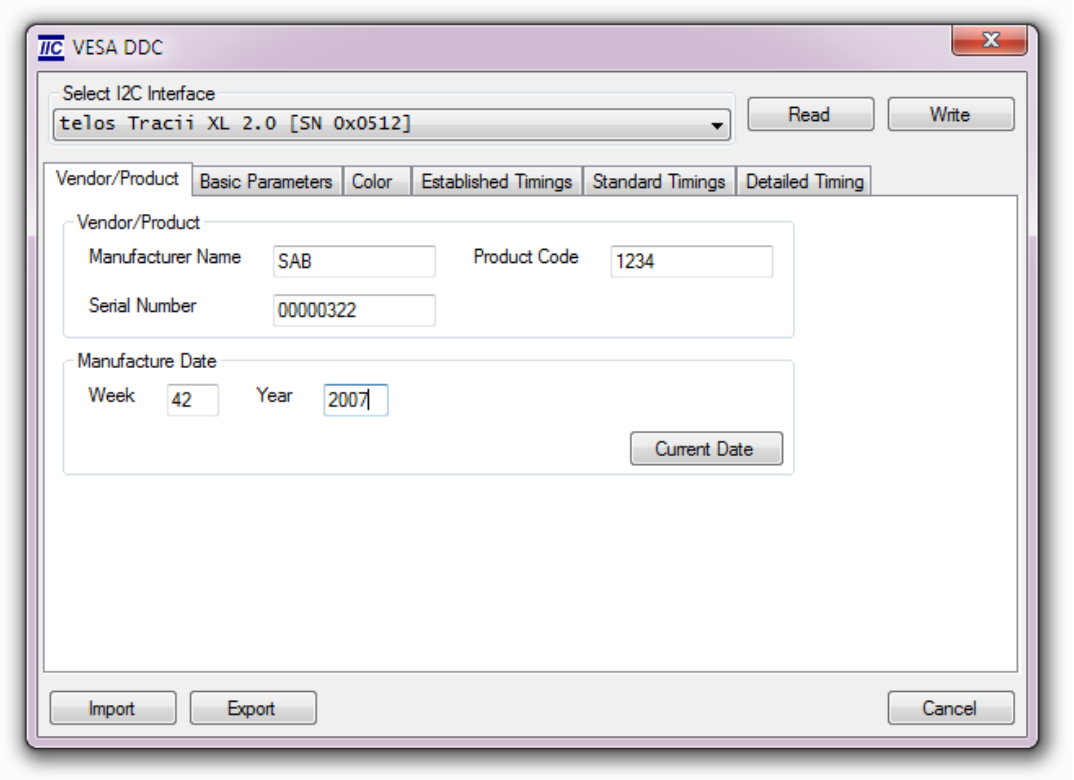


Figure 9.2: VESA DDC

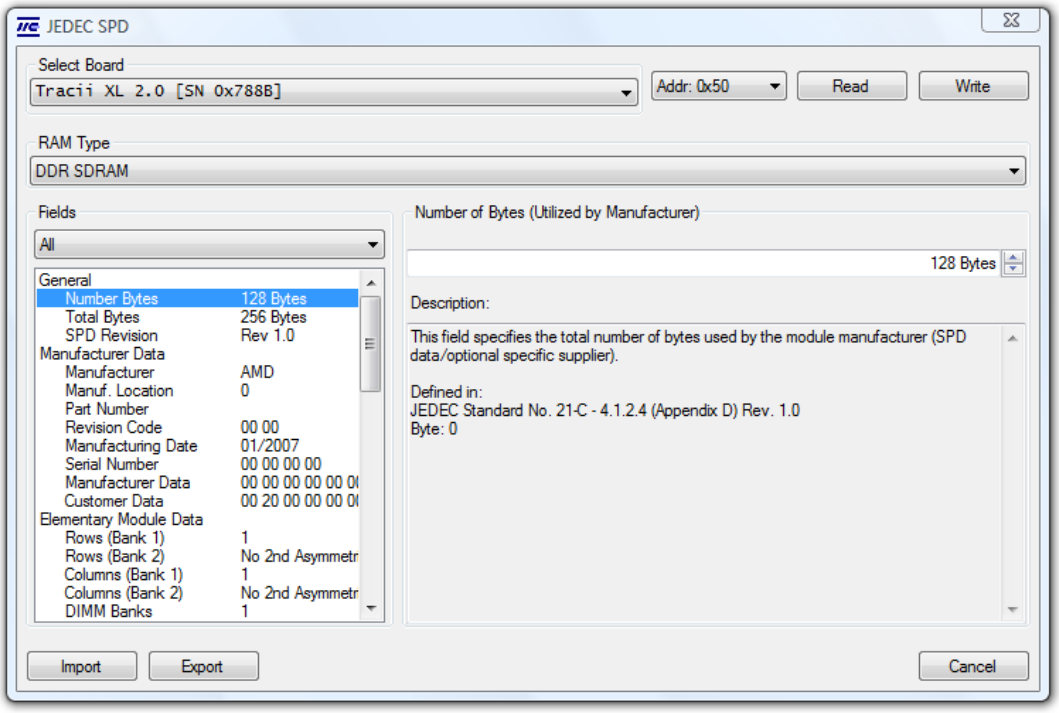


Figure 9.3: JEDEC SPD

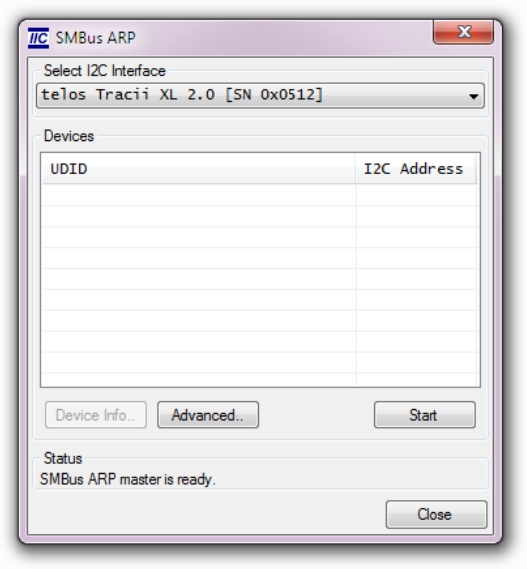


Figure 9.4: SMBus ARP Dialog

Figure 9.3 shows the dialog, which has been opened by clicking onto the menu item Tools|JEDEC SPD.

Like in the VESA DDC dialog there are Read and Write buttons to exchange the configuration with an appropriate EEPROM connected to the I2C bus of the selected I2C interface. In contrast to the VESA DDC dialog the I2C address of the EEPROM is not fix. The user has to specify the I2C address he wants to exchange the data with.

Moreover there are Import and Export buttons, which export the content as an image to a file and reimport it.

All other fields of the dialog serve the modification of the content to be stored in the JEDEC image.

9.4 SMBus Address Resolution Protocol

The SMBus specification version 2.0 describes in section 5.6 an "Address Resolution Protocol" (ARP). Using this protocol a SMBus ARP master can determine the slave address of SMBus ARP slaves, which are using a fix address. Furthermore the master can assign an address to slaves using dynamic addresses during runtime. With the help of this protocol the master can also get some information about the slaves, e.g. the vendor and device IDs.

I2C Studio contains a SMBus ARP dialog, which implements an ARP master. This dialog, opened by the menu item Tools|SMB ARP, is shown in figure 9.4.

Normally the ARP process does not need any interactions of the user. It is simply started by clicking onto the Start button. Afterwards the "Devices" table contains a list of all found devices, which support SMBus ARP. The first column of the table contains the "Unique Device Identifier" (UDID), the second the I2C address used by this device.

The UDID contains some information about the device. An interpreted view of this value can be viewed by selecting a row in the table and clicking onto the Device Info button. A new dialog opens, which is shown in figure 9.5.

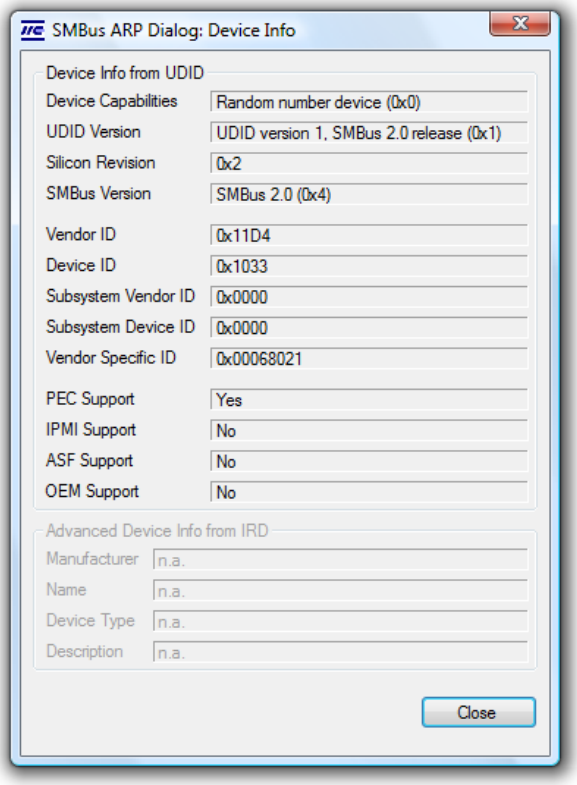


Figure 9.5: SMBus ARP Dialog: Device Info

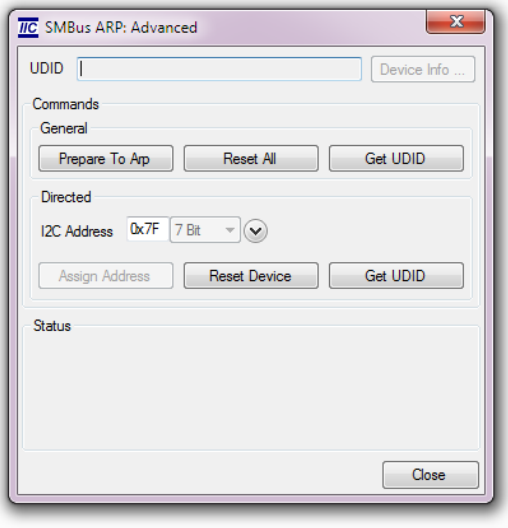


Figure 9.6: SMBus ARP Dialog: Advanced

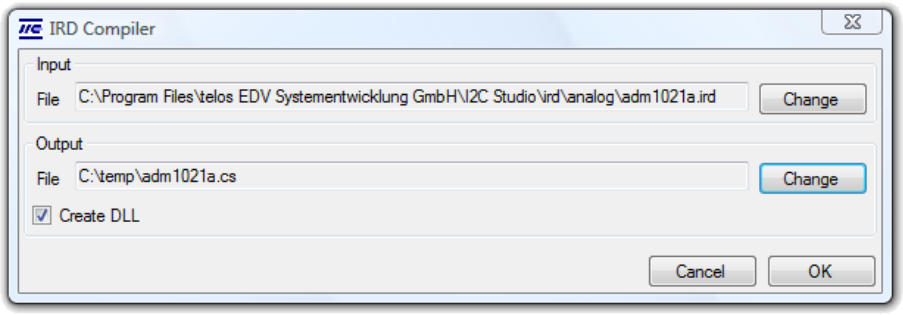


Figure 9.7: IRD Compiler Dialog

When developing a new SMBus slave device, it can be useful to execute the ARP process step by step. For this purpose there is an advanced ARP dialog, which can be opened using the Advanced button. The advanced ARP dialog is shown in figure 9.6.

The meaning of the ARP commands, which can be triggered with the buttons of this dialog, are described in the SMBus specification.

Note: At the moment there are not a lot of SMBus devices on the market supporting SMBus ARP. Even these devices does not work correctly, because their algorithm to calculate the "Packet Error Checking" (PEC) does not conform to the SMBus specification.

Besides the SMBus ARP dialog I2C Studio also supports ARP in the tracer function window. Using a supplied IRD file, which must be registered to the I2C address 0x61, the ARP communication between the SMBus ARP master and the ARP slaves can be viewed.

9.5 IRD Compiler

IRD files are used by I2C Studio to describe the registers and values of an I2C device. This file format is based on XML. I2C Studio comes with a lot of ready to use IRD files for widely used I2C slaves. The user has also the possibility to write its own IRD files.

Using the IRD compiler it is possible to convert an IRD file to a C# source code file. Such a source code file contains one class, which offers a type-safe access to the registers/values of a device via the I2C bus.

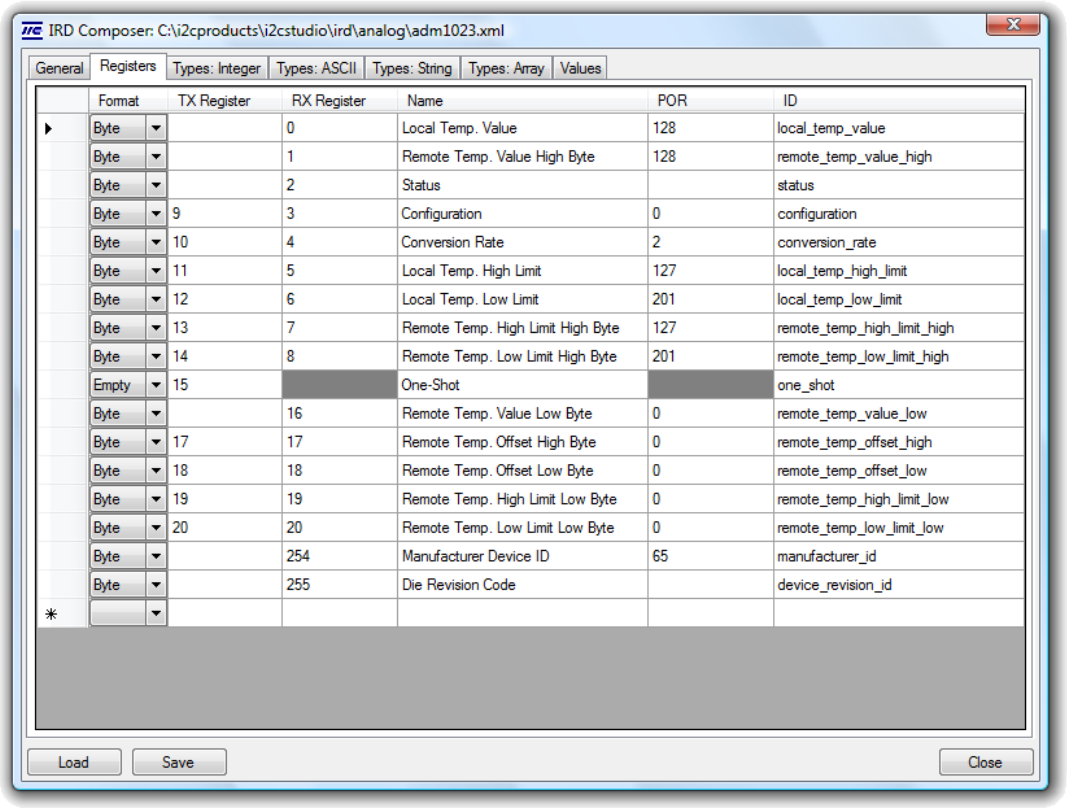
Using such a class communicating with an I2C slave from a script or one's own .NET program becomes a lot of easier. It is no longer necessary to calculate the data bytes to be sent to the I2C slave from its data-sheet. Instead, the developer can simply assign the needed value to a property of the created class.

This dialog offers the following options:

Input File Contains the name of the IRD file, which should be converted.

Output File The C# source code file gets written to a file with this name.

Create DLL If this box is checked, the C# source code gets compiled to a .NET assembly DLL, which can be used from all programming languages which come with .NET support.



	Format	TX Register	RX Register	Name	POR	ID
▸	Byte		0	Local Temp. Value	128	local_temp_value
	Byte		1	Remote Temp. Value High Byte	128	remote_temp_value_high
	Byte		2	Status		status
	Byte	9	3	Configuration	0	configuration
	Byte	10	4	Conversion Rate	2	conversion_rate
	Byte	11	5	Local Temp. High Limit	127	local_temp_high_limit
	Byte	12	6	Local Temp. Low Limit	201	local_temp_low_limit
	Byte	13	7	Remote Temp. High Limit High Byte	127	remote_temp_high_limit_high
	Byte	14	8	Remote Temp. Low Limit High Byte	201	remote_temp_low_limit_high
	Empty	15		One-Shot		one_shot
	Byte		16	Remote Temp. Value Low Byte	0	remote_temp_value_low
	Byte	17	17	Remote Temp. Offset High Byte	0	remote_temp_offset_high
	Byte	18	18	Remote Temp. Offset Low Byte	0	remote_temp_offset_low
	Byte	19	19	Remote Temp. High Limit Low Byte	0	remote_temp_high_limit_low
	Byte	20	20	Remote Temp. Low Limit Low Byte	0	remote_temp_low_limit_low
	Byte		254	Manufacturer Device ID	65	manufacturer_id
	Byte		255	Die Revision Code		device_revision_id
*						

Figure 9.8: IRD Composer Dialog

The conversion is started by clicking onto the OK button.

The .NET assembly `i2capi_dotnet_ird.dll` contains the classes for all IRD files which come with I2C Studio. It can be found in the installation directory of I2C Studio in the subdirectory `lib\`.

9.6 IRD Composer

I2C Studio comes with a large collection of IRD files for all common I2C devices. However, there will be always I2C devices for which there are no ready-to-use IRD files. In such a situation the user has two possibilities for writing his own IRD files: he can write the IRD files, which are in fact XML files, by hand using the "IRD User Manual" and a text editor. Or he can use the IRD Composer to create the IRD files.

Figure 9.8 shows the IRD Composer, which can be opened using the menu item Tools|IRD Composer. This dialog offers a graphical user interface for creating IRD files.

It consists of a collection of tab pages, which contain input masks for defining general information, registers, types, and values. The exact meanings of all these fields are not covered by this manual. They are described in the "IRD User Manual".

After all information for one I2C device has been entered, the Save button can be used to create an IRD file. This file uses, like all IRD files created by the user, the suffix `*.xml` instead of `*.ird`.

The Load button loads an IRD file, which has been created by the user, into the dialog. It is not possible to load IRD files into this dialog, which are part of I2C Studio.

10 License & Support

10.1 License Management

10.1.1 Service Contracts

Each I2C interface bought from telos has got a service contract. This service contract is valid for 12 months. During this time the user gets free software updates of I2C Studio and free support.

To benefit from new features in future release or make use of the support after these 12 months, the service contract can be extended or renewed. To get prices please visit the telos home page or get in contact with our sales department:

```
http://www.telos.info/Support_Policy.352.0.html
```

Service contract extensions are delivered in the form of registration codes.

10.1.2 Upgrade Features

Some I2C interfaces like e.g. telos Tracii XL 2.0 have got several features, which can be bought separately. If the user e.g. owns a basic telos Tracii XL 2.0, he can buy the "Tracer Option" at a later time.

To get this new feature, it is not necessary to send the I2C interface back to telos. Instead the user simply gets a registration code, which will enable the feature in the I2C interface.

10.1.3 Registration Dialog

The registration dialog has got two purposes. It can be used to view the currently available features and the service contracts of an I2C interface. Furthermore the user can enter registration codes, which have be purchased from telos, to extend the service contracts or to enable new features.

The file menu item File|Registration is used to open this dialog. The dialog can be divided into the following parts:

Select Board Contains a list of all I2C interfaces, which are currently visible for I2C Studio. The user has to select the I2C interface, whose licenses should be viewed or modified.

Licenses This table contains the licenses, which are currently stored in the selected I2C interface:

Status Indicates, whether this feature can be used with this version of I2C Studio.

license available
license not available

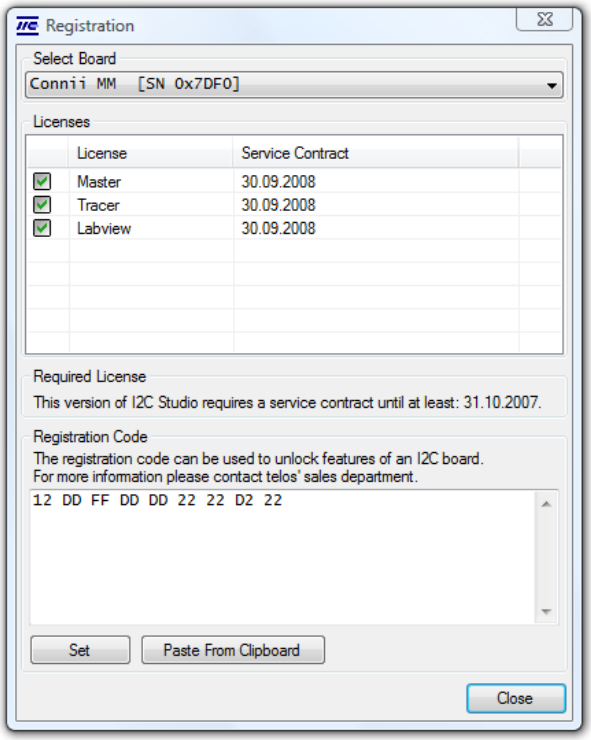


Figure 10.1: Registration Dialog

License Name of the feature.

Service Contract This feature can be used with all I2C Studio versions, which have been release not later than this date. The I2C interface shown in figure 10.1 can be used with all I2C Studio versions released not later than 2008/09/30.

Required License Shows the date, when the currently running version of I2C Studio has been released.

Registration Code One or more registration codes can be entered in this field to enable new features or to renew the service contracts. When a registration code is set using the Set button, the updated license information is stored in the I2C interface itself. So afterwards the registration code is no longer needed.

Note: If the user buys some additional features or a service contract extension together with the basic hardware, all needed license information is already stored in the I2C interface. So the user does not have to enter any registration codes.

10.2 Support

telos offers support for its products by e-mail or phone. We do our best to solve any problem within one working day.

To get into contact with the telos product support:

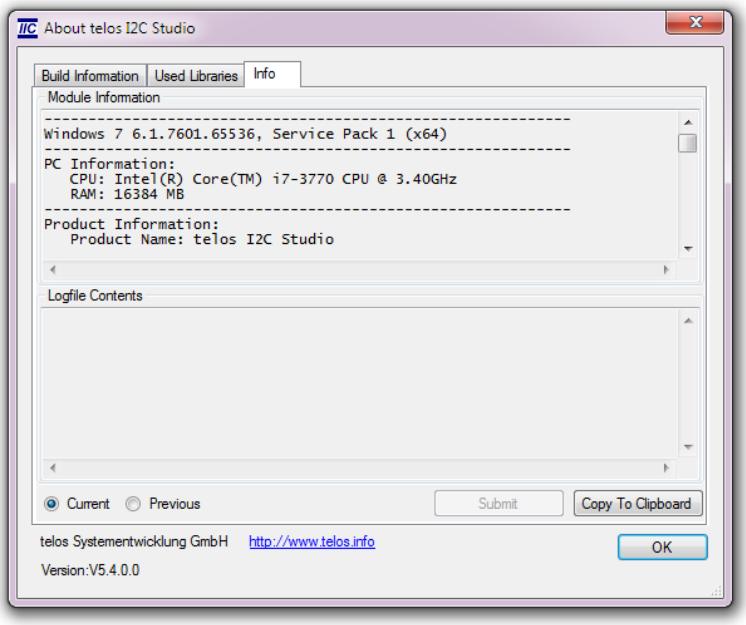


Figure 10.2: About Dialog

telos Systementwicklung GmbH
Schlueterstrasse 16
20146 Hamburg
Germany

Phone: +49 (0)40 450173 61
Fax: +49 (0)40 450173 99
E-Mail: support-i2cstudio@telos.info

Please always add the type and the serial number of your I2C interface to your e-mails. The easiest way to get this information is to use the "About" dialog, which can be opened using Help|About. With the Copy to Clipboard button all needed information is copied to the MS Windows clipboard. Alternatively the Submit button can be used to mail all this information directly to the product support.

10.3 Newsletter

There is a newsletter, which can be subscribed to get always the latest news about I2C Studio and the I2C products from telos:

`http://www.telos.info/i2cnewsletter/`

At this address the user also finds an archive of the newsletters from the last years.

11 Redistribution

11.1 Overview

Sometimes a user wants to distribute an application, which has been written using one of the I2C APIs, to other users. Therefore it is necessary to redistribute the infrastructure to these users, which is needed to communicate with the I2C interfaces from telos.

If the redistribution of the complete I2C Studio is not the desired solution, it is possible to include only the really needed files into the user's own setup package. telos does not offer any support for such setups.

Normally three components are needed: the driver, the I2C scheduler, and the API itself.

11.2 Driver

The driver can be found in the installation directory:

```
<install dir>\drivers
```

If the tools, which are used to create the setup package, do not support the installation of drivers by itself, telos recommends to use the "Driver Install Frameworks (DIFx)" from Microsoft, which is part of the WDK.

11.3 I2C Scheduler

The I2C Scheduler is located in the following executable:

```
<install dir>\bin\tracii_scheduler.exe
```

This executable must be registered as Windows Service during the installation of the setup package. Before a setup package gets deinstalled the Windows Service must be deregistered.

The I2C scheduler offers some command line options to register and deregister itself:

```
tracii_scheduler.exe <option>

  /install_2000      register the I2C scheduler (>= Win2k)
  /remove_2000       deregister the I2C scheduler (>= Win2k)
```

After a successful registration the I2C Scheduler can be found in the Services dialog of the Administrative Tools (see figure 11.1).

11.4 API

Depending on the used API different files must be included into the user's own setup package.

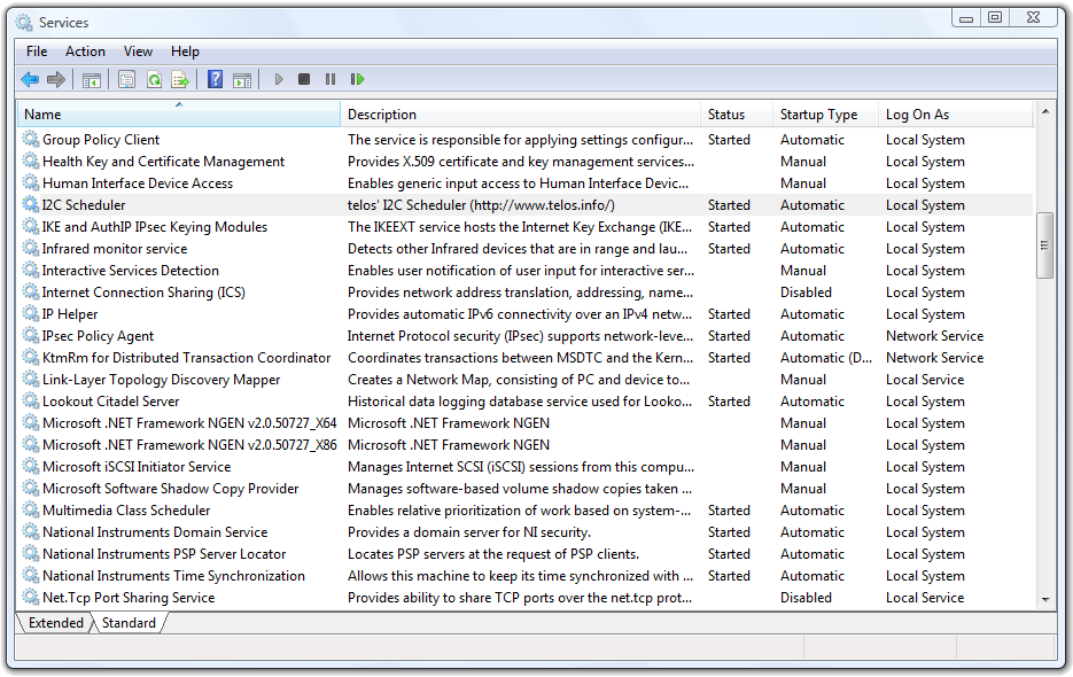


Figure 11.1: Control Panel\Administrative Tools\Services

11.4.1 C

The I2C.C API is located in the following file:

```
<windows dir>\i2capi_c.dll
```

11.4.2 C++

As the I2C.C++ API gets linked statically to the user's application; no additional files must be included into the setup package.

11.4.3 .NET

The I2C.NET API can be found in the following directory:

```
<install dir>\lib\dotnet
```

There are two subdirectories, which contain the 32-bit and the 64-bit versions of the I2C.NET assembly. The interface of both assemblies is compatible.

11.4.4 Java

The JAR file and the DLLs of the I2C.Java API can be found in the following directory:

```
<install dir>\java
```

11.4.5 Labview

For the I2C.Labview API some more files are needed. At first two .NET assemblies are needed: the I2C.NET API and `i2capi_dotnet_labview.dll`. The second assembly can be found in the Assembly Cache of Windows, which can be accessed using the following virtual directory:

```
<windows dir>\assembly
```

In addition to these assemblies it is necessary to add all used I2C VIs to the setup package. These VIs can be found in the following directory:

```
<install dir>\labview
```

The VIs support Labview starting with version 8.2.